VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Inovativní integrace kamer do chytrého předního nárazníku

Innovative Cameras Integration into the Smart Front Bumper

Jan Retych

Diplomová práce

Vedoucí práce: doc. Ing. Jaromír Konečný, Ph.D.

Ostrava, 2021

Abstrakt

Cílem diplomové práce je vytvoření předvývojového subsystému počítačového vidění přední části vozidla za pomocí tří kamer a výpočetní techniky. Součástí práce je realizace živého přenosu ze tří kamer, vyhodnocování panoramatických snímků a prezentace výsledků webovým rozhraním. Subsystém je součástí předního chytrého nárazníků *SmartFace* společnosti *Hella*. Pro demonstraci předního nárazníku vznikl stojan z hliníkových profilů reprezentující umístění senzorů v přední části vozidla.

Klíčová slova

Mikrokontrolér; Kamera; Sběr a zpracování dat; Zpracování obrazu; Jetson Nano; Panorama; Předvývoj; *C++*; Multiplatformní; Inovace

Abstract

The aim of the diploma thesis is to create a pre-development subsystem of computer vision of the front part of the vehicle including three cameras and computer technology. Part of the work is the implementation of a live broadcast from three cameras, evaluation of panoramic images and presenting results via a web interface. The subsystem is part of the smart front bumper named *SmartFace* of the *Hella* company. To demonstrate the front bumper, a stand has been made from aluminum profiles to represent the location of sensors in the front of the vehicle.

Keywords

Microcontroller; Camera; Data collection and processing; Image processing; Jetson Nano; Panorama; Pre-development; *C++*; Multiplatform; Inovation

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce doc. Ing. Jaromíru Konečnému, Ph.D. za poskytnuté konzultace, možnost výběru zadání a za důvěru při realizaci projektu. Za příležitost podílet se na předvývojovém projektu chytrého nárazníků *SmartFace* bych rád poděkoval společnosti *Hella*. Poděkování rovněž patří Ing. Zdeňkovi Slaninovi, Ph.D. za poskytnutí senzorů a výpočetní techniky pro testovací potřeby. A za kontrolu textu bych chtěl poděkovat PhDr. Heleně Retychové.

Obsah

Seznam použitých zkratek a symbolů

AVR	–	Advanced Virtual Risc
ASIL	–	Automotive Safety Integrity Level
ASIC	–	Application Specific Integrated Circuit
ADC	–	Analog-to-Digital Converter
ALDL	–	Assembly Line Diagnostic Link
CAN	–	Controller Area Network
CPU	–	Central Processing Unit
CMOS	–	Complementary Metal Oxide Semiconductor
CSI	–	Camera Serial Interface
ECU	–	Electronic Control Unit
EEPROM	–	Electrically Erasable Programmable Read-Only Memory
EU	–	European Union
FPGA	–	Field Programmable Gate Array
FPS	–	Frames per second
FPU	–	Floating-Point Unit
GPIO	–	General-purpose input/output
HTTP	–	Hypertext Transfer Protocol
ID	–	Identification
JIT	–	Just In Time
LVDT	–	Linear variable differential transformer
LDA	–	Linear discriminant analysis
LDF	–	Local Interconnect Network Description File
LIN	–	Local Interconnect Network
LoG	–	Laplaceův Gaussioův filtr
LSH	–	Locality Sensitive Hashing
MIPS	–	Million Instructions Per Second
MOST	–	Media Oriented Systems Transport
MCU	–	Microcontroler

OBD	–	On-Board Diagnostics
OS	–	Operační systém
PCA	–	Principal Component Analysis
PIC	–	Peripheral Interface Microcontroller
PWM	–	Pulse Width Modulation
RTR	–	Remote Transmission Request
RFID	–	Radio-Frequency Identification
RPM	–	Revolutions Per Minute
RGB	–	Red-Green-Blue
RMS	–	Root Mean Square
SPI	–	Serial Peripheral Interface
TDMA	–	Time Division Multiple Access
UART	–	Universal Asynchronous Receiver – Transmitter
UI	–	User interface
USB	–	Universal Serial Bus
UTP	–	Unshielded twisted pair
UML	–	Unified Modeling Language
VHDL	–	VHSIC Hardware Description Language

Seznam obrázků

Seznam výpisů zdrojového kódu

Úvod

Zadání diplomové práce vzniklo potřebou předvývoje kamerového subsystému pro chytrý přední nárazník jménem *SmartFace* v rámci mezinárodní spolupráce s pobočkou v *Berlíně* společného podniku *Hella Plastic Omnium*. Nárazník nabízí pro budoucí trh různé elektronické systémy, senzory (parkovací senzory, kamery, lidar, radar apod.) a světlomety v jednom celku a současně umožňuje jednoduchou montáž. V rámci diplomové práce je realizován stojan pro uchycení senzorů a box pro umístění výpočetní techniky. Stojan reprezentuje šířku předního nárazníků a je vytvořen jako jeho náhrada pro pobočku v *Ostravě*. Úkolem je vytvoření kamerového subsystému pro chytrý nárazník poskytující panoramatické snímky ze tří kamer. Pro tyto účely je vytvořena obslužná aplikace.

Předvývojový projekt chytrého nárazníků *SmartFace* sestává z různých subsystémů. Jedním z nich je kamerový subsystém, který je z hlediska komunikace a softwarové části nejnáročnější. Vzniklý obslužný software, stojan a box slouží pro demonstrační účely a jako základní kámen pro další budoucí předvývoj či vývoj předního nárazníku *SmartFace*. Zřetel je brán zejména na přenositelnost systému mezi platformami a operačním systémem *Linux* a *Windows*. Důležitým faktorem je rovněž libovolná zaměnitelnost senzorů a výpočetní techniky. Stojan obsahuje dvě polohovatelná ramena o rozpětí až 1945 mm, sestavená z hliníkových profilů, nabízející jednoduchou montáž či demontáž dalších dílů, senzorů nebo celých systémů. Obslužný software poskytuje živý přenos z kamer a panoramatické snímky z aktuálních dat. Pro generované panorama je prioritou dodržení včasného vyhodnocení. Kvalita vyhodnocení a rozdílné přechody mezi snímky již tak podstatné nejsou. Nabyté znalosti o zpracování obrazu jsou řádně zdokumentovány a je využito UML diagramů.

V první kapitole jsou popsány předpoklady pro výběr mikrokontroléru, srovnání s automobilovým průmyslem, popsána odlišnost oproti běžné aplikaci a jsou uvedeny příklady nejčastějších mikrokontrolérů v automobilovém průmyslu. Pro elektroniku v automobilovém průmyslu platí stejná pravidla jako pro běžnou elektroniku v domácnosti, požadavky na výstupní produkt jsou však často mnohem přísnější z hlediska bezpečnosti. Ty jsou často specifikovány jak zákazníkem, tak i dodavatelem, a vyplývají z různých bezpečnostních analýz. Pro potřeby automobilového průmyslu jsou k dispozici stovky typů mikrokontrolérů již připravené pro specifický systém automobilu, jako je například systém ABS, adaptivní podvozek nebo systémy řízení motoru dodávané například výrobci *Infineon Technologies*, *NXP*, *Renesas* a *Texas instruments*.

Popis dostupných kamerových systémů, jejich vlastnosti a cílová aplikace je popsána v kapitole druhé. Kamerové systémy můžou být tvořeny vícero kamerami a dalšími senzory. S modernizováním se stává počítačová vize součástí každého automobilu poskytující pohodlí a bezpečnost. Různí asistenti jsou vítanými pomocníky při parkování, couvání, zabránění nehodám, srážce se zvířaty či s chodci. S pomocí kamerových systémů je vytvářeno rovněž i autonomní řízení.

V kapitole třetí jsou popsána různá komunikační rozhraní používaná v automobilech, jejich možnosti a komunikační principy. Systém elektroniky moderního automobilu je z důvodu vzrůstající náročnosti tvořen subsystémy rozdělenými na základě bezpečnostních úrovní, spolehlivosti a komunikační náročnosti. Subsystémy mezi sebou často komunikují skrze prostředníky. Nejrozšířenějším rozhraním je rozhraní *CAN* pro lokální využití, pro složitější systémy je však již voleno rozhraní *FlexRay* nebo *MOST*. Uplatnění nalezne i starší rozhraní *LIN*, kde případné chyby zpráv nemají vliv na bezpečnost okolí.

Úpravě a zpracování snímků programováním za pomoci knihovny *OpenCV* se zaměřuje čtvrtá kapitola. V souvislosti s cílem poskytovat panoramatický snímek jsou uvedeny způsoby k jeho získání sestávající se ze 4 kroků. Jako první je zapotřebí získání klíčových bodů a deskriptorů popisující objekty ve snímcích. Následuje porovnání deskriptorů mezi dvěma snímky a získání shod. Následuje filtrace shod. V kapitole jsou popsány různé dostupné metody pro získání klíčových bodů a deskriptorů, jakými jsou *Harrisova a Shi-Tomasova detekce rohů*, metody *FAST*, *BRIEF*, *SIFT*, *SURF* a metoda *ORB*. Pro porovnání shod mezi snímky jsou popsány metody *FLANN* a *BruteForce*. V poslední řadě je popsáno samotné spojení snímků k získání panoramatu.

Pátá kapitola se věnuje požadavkům na kamerový subsystém, popsání vzniklých podmínek pro výběr kamer a mikrokontroléru, provedeném testování a realizaci demonstračního stojanu a podstav. Podmínky vzešly na základě požadavků a provedeném testování. Subsystém musí splňovat požadavky, jako je přenositelnost, poskytovat živý přenos snímků z kamer a vyhodnocovat panorama. Subsystém se skládá až ze tří kamer. Pro demonstrační účely a fyzickou prezentaci je vytvořen stojan s boxem o rozpětí předního nárazníku s možností libovolné montáže senzorů. Pro vývoj obslužného softwaru jsou vytvořeny stolní podstavy kamer. V kapitole jsou rovněž popsány fyzické vlastnosti stojanu a možnosti flexibility hliníkových profilů stojanu.

Vytvořený obslužný software skládající se ze tří nezávisle na sobě fungujících aplikací je popsán v kapitole šestá. Rozdělení aplikací je dle platforem a úkolů. Úkoly obslužného softwaru jsou sběr a distribuce živých snímků z kamer, vytvoření a distribuce panoramatu a prezentování výsledků webovým rozhraním. Primárním zvoleným programovacím jazykem je *c++*. Pro zpracování panoramatu je využita knihovna *OpenCV* a pro komunikaci pomocí HTTP protokolu, správu webového serveru a komunikace pomocí websocketu je využito knihoven *Mongoose* a *POCO*. Vytvořené aplikace jsou vícevláknové a jsou spuštěny na odlišných platformách a operačních systémech.

Poslední sedmá kapitola se věnuje testování výsledného obslužného softwaru. Je provedeno zpětné testování a určení kvality vyhodnoceného panoramatického snímku a testování živého pře-

nosu obrazu z kamer. Určení kvality panoramatu je vyjádřeno hodnotou mediánu a RMS. Vyhotovené testy různých kvalit panoramatů jsou rovněž k dispozici v příloze B, C, D a E.

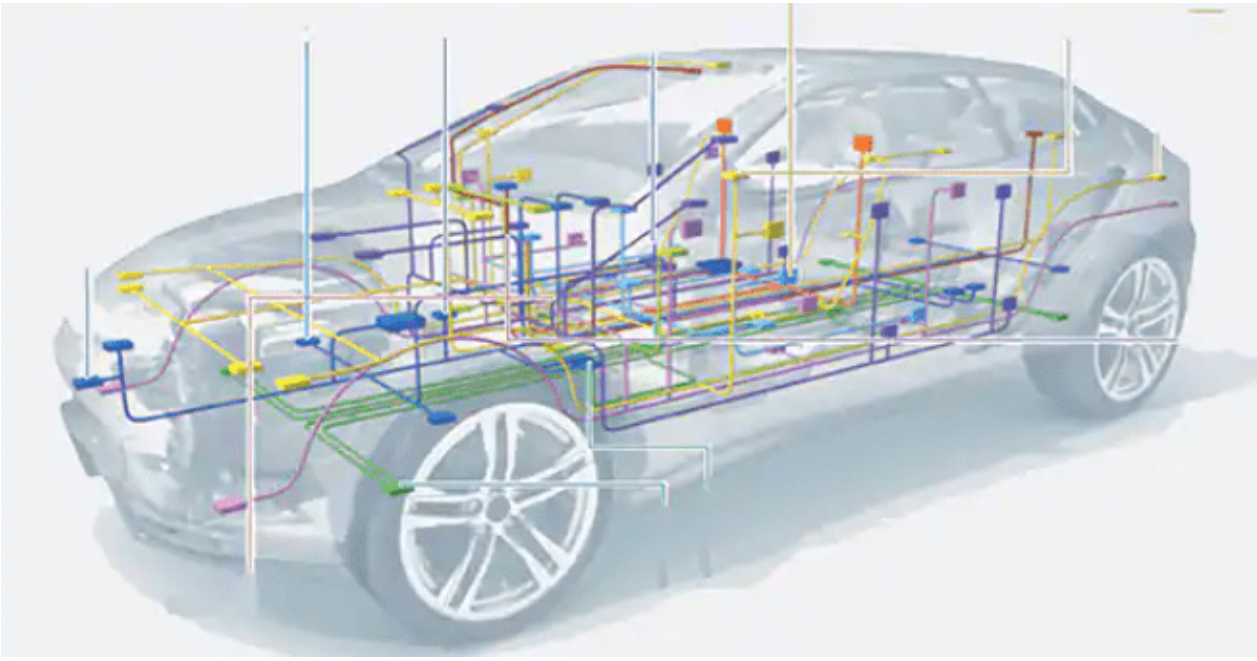
Kapitola 1

Mikrokontroléry pro automobilový průmysl

K základním aspektům použití mikrokontroléru jsou jeho dispozice a možnosti. Typicky v první fázi výběru rozhodují zejména funkce, typy periferií, výkon a napájecí možnosti. Ve druhé fázi výběru by mělo dojít k hlubšímu průzkumu vlastností MCU, jako jsou například fyzické realizace sběrnic, jejich možnosti a vliv okolní teploty na funkčnost. V poslední fázi přichází na řadu bezpečnost, spolehlivost kontroléru a určení pravděpodobnosti selhání. V automobilovém průmyslu a i v jiných je však výběr prováděn přesně opačně. Nejdříve je zapotřebí zajistit bezpečnost, poté určit podmínky použití a následně přejít k výběru a poptávce. Pro předvývoj není nutno počítat s bezpečností, neboť se jedná pouze o demonstraci možného budoucího řešení. V následující kapitole bude popsán výběr mikrokontroléru obecně a pro automobil, dále jsou uvedeny příklady některých MCU pro automobilový průmysl.

Mikrokontroléry či ovládací jednotky jsou využívány k ovládání jakékoliv elektroniky. Od odemykání zámků, stahování okének, předávání informací z čidel až k řízení ochranných systému, *ABS*, řízení vstřikovaní a podobně. Několik let zpátky – dnes součást základní výbavy – obsahovalo jen velmi malé množství vozidel elektroniku. Ve starších vozidlech v letech mezi 1990 až 2000, byla většina komponent řešena stále mechanicky. Počet MCU či ECU bylo možné nalézt v řádech jednotek až desítek. Záleželo na výbavě automobilu. Cena elektroniky byla výrazně vyšší, a tak se v základní výbavě vyskytovalo opravdu jen malé množství MCU v řádech jednotek. V takovém automobilu byla pouze jediná hlavní řídící jednotka, do které byly zapojeny všechny senzory a spínací prvky. V těchto letech se však začaly vyrábět i luxusní vozy, obsahující velké množství elektroniky. Lze hovořit o počátku modernizace automobilů. Vozidla střední třídy obsahovala 20 až 50 jednotek, luxusní vozy již 70 až 110. Technologicky bylo velmi složité zůstat u jedné kontrolní jednotky a systém se začal členit na části a hojně využívat komunikace po sběrnici *CAN*. V kontrastu s dnešní dobou, kdy je v automobilu nespočet jednotek mezi sebou komunikujících, se komunikační sběrnice začínaly zahlcovat a byly přetěžovány. Proto dochází k rozdělení systému na sub-systémy komunikující na oddělených sběrnicích. V současnosti je možné nalézt 100 a více jednotek v každém moderním voze. Při takovém množství bylo zapotřebí, aby sub-systémy komunikovaly nezávisle na sobě

a v případě nutnosti komunikovat mezi sebou pak využili prostředníka. Při rozdělování sub-systému dochází i k rozdělení podle bezpečnosti pro provoz vozidla a jejich podmínek komunikace. Systém s kritickým označením a tedy fatálními následky v případě selhání komunikuje nepřetržitě, i když se jedná o stále stejnou informaci, musí být v daném časovém intervalu obnovovaná. Lze hovořit o vytvoření systémů reálného času. V případě selhání senzoru či přípravku, který o sobě nedává vědět, lze ostatními posluchači na sběrnici detekovat, že došlo k selhání, a přejít ke krizovému řešení, jako je například rozsvícení palubní kontrolky, omezení výkonu, v horším případě pak vypnutí motoru a zastavení vozidla. Na obrázku 1.1 lze vidět barevně odlišeny sub-systémy moderního automobilu. [2, 3]



Obrázek 1.1: Jednotlivé sub-systémy automobilu komunikující nezávisle na sobě

1.1 Předpoklady pro výběr MCU a ECU v automobilovém průmyslu

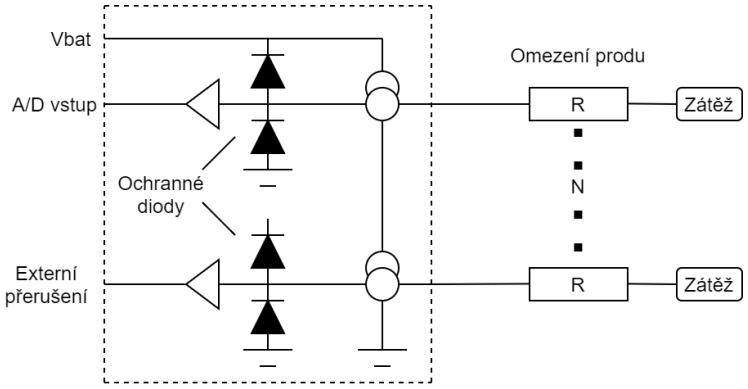
MCU je volena podle aplikace, ta může mít při poruše žádné či banální následky, jsou však i systémy řízené pomocí MCU, které musí i po poruše fungovat správně, nebo začít opětovně fungovat správně a nebo se ukončit, aby neohrozily svým nesprávným zásahem jiný systém. Pro realizaci elektroniky v automobilu platí stejná pravidla a normy jako pro každý jiný spotřebič v domácnosti, tedy musí v Evropské unii splňovat standardizovanou normu *CE*. Velkým faktorem je pak dopad systému na bezpečnost vozidla, osob a okolí, které rozhodují o funkčnosti elektroniky. Další požadavky většinou tvoří automobilky, ku příkladu stanovení odolnosti proti elektromagnetickému záření vyššímu, než je specifikováno normou, udržení funkcionality u vyšších teplot (okolo 90 °C)

a podobné běžné případy vyplývající ze zkušeností provozu a laboratorních testů a finální destinace produktu. [2, 3, 4]

Při výběru MCU vhodného pro automobil si lze povšimnou, že mnohdy výrobce specifikuje danou jednotku jako vhodnou pro automobilový průmysl. Znamená to, že jednotky splňují běžné požadavky k jejich aplikacím. Nicméně je vhodné vždy zkontrolovat všechna kritéria. Obecně lze popsat, že nálepka „vhodný pro automobilový průmysl" znamená splnění teplotních rozmezí $-40\,°C$ až $120\,°C$, možnost komunikovat skrze rozhraní *LIN*, případně *CAN* a navýšení hodnoty napětí pro logickou úroveň 1 z důvodů uzemnění vozidla. Uzemnění mnohdy tvoří smyčky dlouhé několik metrů a vzniká šum při procesech vyžadující vysoké proudy. Komunikace skrze *LIN/CAN* se však dá realizovat i externím zařízením, překladačem, který přijímá zprávy skrze jinou komunikační periferii, například skrze I^2C. Požadavek tedy na komunikaci pomocí *LIN/CAN* není nezbytný, nicméně při sériové výrobě je výhodnější použít jedno zařízení obsahující vše, i kdyby bylo o něco dražší, než dvě různá. U MCU se lze setkat často i s požadavkem na funkční bezpečnost. Příkladem může být tzv. „lock-step" provádění operací, detekování chyb pamětí *FLASH*, *EEPROM* i *RAM*, zamezení přístupu k datům programu, monitorování FPU, monitoring hladin napětí, resetování watchdogu v případě selhání a další. Dle aplikace a požadavků se pak jedná spíše o samotné funkce jednotky. Mezi časté patří funkce detekce nízké úrovně napětí, která zajistí bezpečný restart jednotky a její opětovnou funkčnost. Dále analogově-digitální převodník sloužící ke komunikaci s analogovými senzory. Běžnou funkcí automobilových MCU bývá také přivedení vstupního napětí baterie přímo na výstup, často procházející skrze odpor pro regulaci maximálního protékajícího proudu (Obrázek 1.2). Případně potenciometr řízený jednotkou, tvořící napěťový dělič, pro regulaci napětí výstupního pinu. [5, 4]

ECU je většinou jednoduchá řídící struktura, aby však nebyla jeho podstata příliš jednočará, obsahuje několik funkcí, které lze přepínat odesláním definovaných pulsů na jeden či více vstupních pinů. Funkce již specifické pro svou aplikaci jsou pak například detekce zastavení krokového motoru a čítač pozice a otáčení sloužící k ušetření softwarového výkonu a k vyhodnocení rychlosti rotace a směru. Tato funkce má své uplatnění zejména pro otočné navigační kolečka ovládající audio, video a navigaci. Detekce zastavení krokového motoru šetří nárok na zpětnou vazbu skrze senzor, který by udával stav krokového motoru. Dostupnou vlastností pro MCU bývá rozdělení *EEPROM* pamětí na dvě části. Do jedné je zapisován program a data a ve druhé jsou uložené základní instrukce pro nebezpečné a rizikové stavy. Po výběru vhodné jednotky pro danou aplikaci je však brána v potaz bezpečnost a k takřka každému MCU je implementován tzv. *watchdog* časovač. Jeho funkcionalita spočívá v oscilaci a v případě přechodu MCU do stavu, kdy neodpovídá, přivede *watchdog* časovač signál k resetování. V neposlední řadě je zajišťováno vhodné napájení jednotek pomocí filtrace, zeslabení či zesílení a v některých případech i galvanickým oddělením. [5, 6, 4]

Požadavky na elektroniku ve vozidlech nejsou odlišné od požadavků pro jiné elektronické výrobky, jako je například kuchyňský spotřebič. I zde platí norma *CE* vycházející z francouzštiny „conformité européenne", tedy splnění požadavků *EU*. Požadavky nad rámec této normy jsou sou-



Obrázek 1.2: Příklad jednoduché ECU

částí funkční bezpečnosti a bezpečnosti obecně, případně požadavek automobilky samotné kvůli splnění standardu. Nejčastěji se jedná o rozsah teplot, o garanci doručení zpráv po sběrnici nebo realizace spotřebiče co možná energeticky nejekonomičtější cestou – o tomto kritériu však nejčastěji rozhoduje cena, která tento požadavek eliminuje velice rychle. [7]

1.2 Dostupné mikrokontroléry a jejich aplikace

Nabízené MCU pokrývají takřka kompletní poptávku ve všech kategoriích. V případě nedostatku výkonu či nesplnění podmínek na periferie jsou i samotní výrobci ochotní ke změnám v sériové výrobě a přizpůsobit či vyvinout další mikrokontrolér. Příklady aplikací MCU jsou například bezpečnostní aplikace, řízení otáčení kol, elektrický posilovač brzd, kontrola stability vozidla při brzdění, elektrická parkovací brzda, spuštění airbagů, adaptivní podvozek, systémy řízení motoru, řídicí jednotky převodovky, systémy pro správu střídačů a baterií pro hybridní a elektrická vozidla a další.

1.2.1 *Infineon Tri-core MCU*

Infineon Tri-core MCU je nejčastěji používán k regulaci výfukových zplodin a k udržení spotřeby automobilu na co nejmenší úrovni. Byl navržený pro robustnost, výkon a bezpečnost. Typů tohoto mikrokontroléru je hned několik, lze volit od jednoho až po tři nezávislá jádra procesoru. Pro detekci chyb a rychlé odezvy procesů splňující podmínky *ASIL* kategorie D funkční bezpečnosti je využito takzvaně „lockstep" jader, neboli uzamykání kroků. „Lockstep" architektura zajišťuje rozsáhlou sadu bezpečnostních prvků. Architektura procesoru je navržená rovněž tak, aby bylo možné využívat paralelismu úloh bez většího zásahu do softwarové části nebo měnění bezpečnostních opatření. Ku příkladu *AURIX™ TC26xL* 32-bitový *TriCore MCU* se vyznačuje špičkovým výkonem v reálném čase běžící na operační frekvenci 200 MHz v celém rozsahu teplot a zajišťující robustní manipulaci s bity. Obsahuje dva vysoce výkonné 32bitové superskalární procesory *TriCore V1.6.1* s 6/4 stupňovou sběrnicí. Paměť procesoru má rozdělení pro každé jádro zvlášť. Výrobce nabízí mož-

nost alternativních pouzder pro absorbování vyšších teplot, případně přidání chladičů na pouzdro. V neposlední řadě MCU disponuje inovativním napájením jediným vstupem s integrovaným 8bitovým pohotovostním kontrolérem pro režimy extrémně nízké spotřeby. *AURIX™* disponuje dalšími funkcemi: [8, 9]

- Maximální velikost programu až 2,5 MB.
- 240 kB *SRAM*.
- 48× *DMA* kanál.
- Původní *Capture Compare* jednotka (*CCU*) a časovač pro obecné účely (*GPT*).
- Analogově-digitální převodníky Delta-sigma pro rychlé a přesné měření.
- 12bitový A-D převodník *SAR* (5 V / 3,3 V).
- Rozhraní senzorů (*SENT* / *PSI5* / *PSI5S*).
- Vysokorychlostní sériové rozhraní pro komunikaci mezi procesory.
- Ethernet 100 Mbit.
- Modul *FlexRay* se 2 kanály.
- Rozhraní *Micro Second Bus* (*MSC*).
- Asynchronní / synchronní sériová rozhraní (*ASC*, *QSPI*, I^2C).
- Podpora vícejádrového ladění (*MCDS*).
- Integrovaný regulátor napětí s integrovaným pohotovostním regulátorem.
- Celý teplotní rozsah pro automobily −40 °C až 125 °C.
- Možnosti pouzder: *LQFP-144* / *LQFP-176* / *LFBGA-292* / Bare die.

Obrázek 1.3: Pouzdro *MCU TriCore Aurix* [10]

1.2.2 *Atmel AVR* mikrokontroléry

Atmel AVR mikrokontrolér vyniká ve výkonu a flexibilitě pro použití v automobilovém průmyslu. Dokáže posloužit i jako výkoný napájecí prvek. Disponuje *Hardvardskou* architekturou, což umožňuje vyniknout velkým výpočetním výkonem a snížit instrukce na procesorové úrovni. Oproti ostatním

MCU *Atmel AVR* disponuje vestavěným ADC převodníkem, interním oscilátorem a až 6ti sériovými linkami s možností práce i s módem v režimu spánku pro úsporu energie. Tyto MCU jsou hojně využívané i pro vývojářské desky *Arduino*, zejména *ATmega328* a *ATmega2560*. *Atmel* rozděluje MCU na tři kategorie: *TinyAVR*, *MegaAVR* a *AVR XMEGA*. Lze si vybrat od 8bitového procesoru až po 32bitový procesor. Jak z názvu vyplývá, *TinyAVR* jsou MCU s menším vypočetním výkonem, menším počtem periferií a funkcí. *MegaAVR* je rozšíření menší verze o vícero periferií a o větší výpočetní výkon. *XMEGA* kategorie je vhodná pro aplikace reálného času a současně disponující nízkou spotřebou. [8, 11] Příkladem MCU kategorie *XMEGA* je ATxmega128a1. Jedná se o 8 až 16ti bitový *AVR* MCU s až 128 kB programovatelnou *flash* pamětí, 8 MB bootovacím prostorem a 8 kB *SRAM*. Disponuje 4-kanálovým *DMA* kontrolérem, může pracovat až na 8 různých systémových událostech najednou a při taktování 32 MHz dosahuje výkonu 32 *MIPS*. Digitálních komunikačních periferií je k dispozici 8 pomocí rozhraní *UART*, 12 pomocí *SPI* a 4 skrze I^2C. K obsluze okolí MCU nabízí 24 *PWM* výstupů, 24 vstupů s funkcí „capture", tedy zachytit, a dalších 24 s funkcí „compare", tedy porovnat. Pouzdra série A1 obsahují 100 pinů. Tento konkrétní mikrokontrolér lze použít například v průmyslové automatizaci, k ovládání motorů, plošin, k řízení klimatických podmínek prostor, k řízení napájení pomocí přenosných baterií, v komunikacích mezi zařízeními, k přesnému měření nebo v optické a zdravotnické sféře. Doporučené provozní teploty jsou od −40 °C po 85 °C, nicméně *Atmel* nabízí i MCU opět s označením a určením pro *automotive* s odolností i při vyšších stupních například 105, 125 a 150 °C a disponujícím komunikačním rozhraním *LIN/CAN*. MCU splňující požadavky pro automobilový průmysl jsou například *AT90CAN32*, *AT90CAN64* a *AT90CAN128* lišící se velikostí *flash* pamětí, kdy velikost odpovídá konečnému číslu v názvu v kB. Dalšími MCU, které už však postrádají komunikační periferii *CAN*, nicméně zdolávají teplotní hranici 150 °C, jsou *ATtiny861* a *ATmega64M1*. [12, 13]

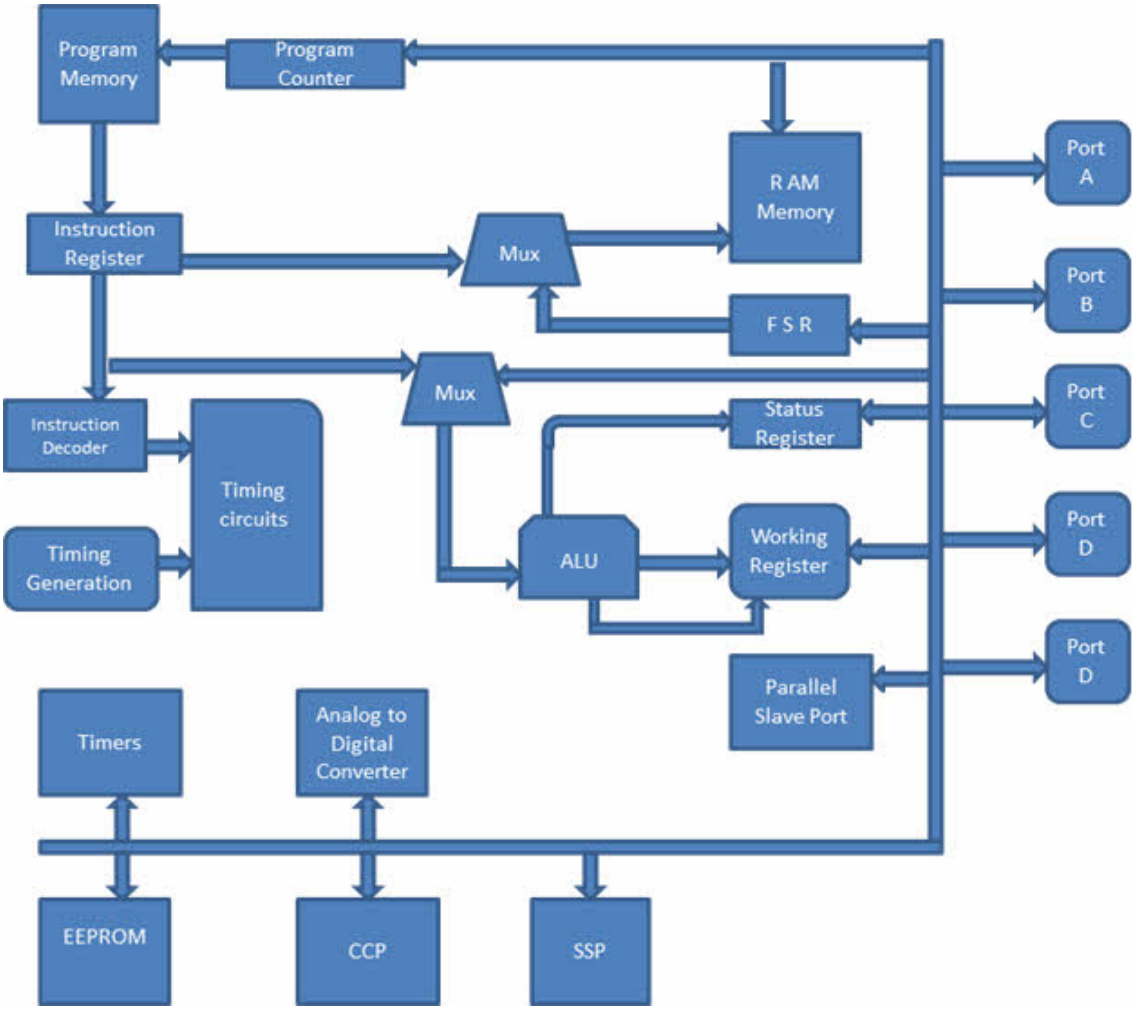
Obrázek 1.4: Pouzdro MCU *Atmel ATxmega128a1* [14]

1.2.3 *PIC* mikrokontroléry

Z anglického slova „peripheral interface microcontroller", tedy mikrokontroléry periferního rozhraní, které jsou schopné zvládat různé úkony a kontrolují generovanou linku. Pro svou jednoduchost a praktičnost jsou *PIC* MCU využívané v chytrých telefonech, automobilech, audiotechnice a také ve zdravotnictví. *PIC18F458* a *PIC18F258* jsou často volenými v automobilovém průmyslu díky zvládnutí komunikace skrze *CAN*, 10bitovém 8 kanálovém ADC modulu, výstupní piny jsou schopny dodat až 25 mA, disponuje dvěma 8 a dvěma 16bitovými časovači. Obsahuje *flash* paměť o velikosti 32 kB, pamět *SRAM* o velikosti4 kB, dosahuje výkonu 10 *MIPS* o frekvenci 40 MHz. Pouzdro obsahuje 40 pinů, z nichž většina má pevně definovaný účel. Volitelných I/O pinů je k dispozici 18. Procesor je *Harvardské* architektury a obsahuje rozdělení pamětí na programovou a datovou. Datová pak obsahuje paměť *RAM* a *EEPROM* sloužící k uchování registrů pro základní operace nebo speciální funkce. Série *PIC16* MCU je sestavená z 5ti různých portů označených písmeny A, B, C, D a E. Pro každé písmeno jsou definované funkcionality, kdy pro port A platí 16ti bitový I/O konektor definován *TRISA* registrem, pro B 8 bitový I/O konektor sloužící navíc jako spínač přerušení. Písmeno C je 8 bitový I/O konektor definován tentokrát *TRISC* registrem. Port D, 8 bitový I/O konektor, plní funkci připojení komunikační sběrnice k procesoru a poslední PORT E, 3 bitový I/O konektor, nabízí možnost A/D převodu signálu. *PIC* MCU obsahují velmi přesné oscilátory. Popsaná architektura je vidět na obrázku 1.5. [8, 15, 16]

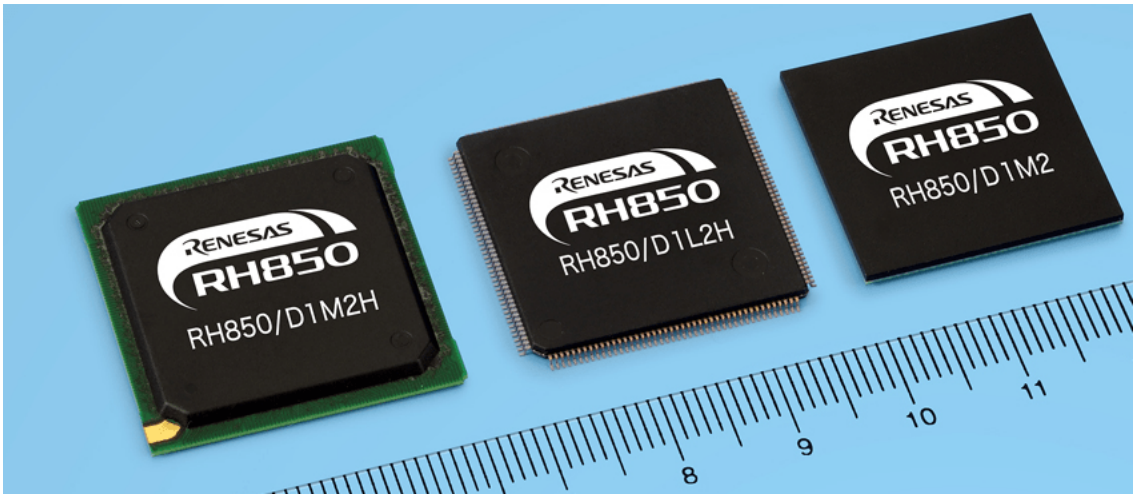
1.2.4 *Renesas* mikrokontroléry

Výrobce *Renesas* se specializuje pro odvětví *automotive* a nabízí celou škálu 32 bitových procesorů s enormním výkonem v kombinaci s malou spotřebou. Samotná Rodina procesorů *RH850* pro *automotive* disponuje řadou vnitřních bezpečnostních funkcí a funkční bezpečnosti pro základní i složitější operace. V nabídce jsou jedno-jádrové, více-jádrové, „lock-step" či kombinované procesorové struktury. Kromě výkonu, velmi malé spotřeby a bezpečnosti je dále kladen důraz na vysokou škálovatelnost, tedy aby v případě záměny MCU mohl být software opět použitelný, například při přejití na základnější MCU, nebo naopak při výběru komplexnějšího MCU. Konkurence schopnou výhodou výrobce *Renesas* je bezpochyby nabízená velikost *flash* pamětí o velikosti až 4 MB a *RAM* pamětí o maximální velikosti až 2 GB. Frekvence procesorů je obvykle 120 MHz nebo 400 MHz. Architektura procesorů je *Harvardská* se sadou instrukcí *CISC* i *RISC*. Největší nabízené pouzdro obsahuje 484 pinů a slouží zejména pro náročnější 2D vykreslování (obrázek 1.6), jako je například simulace provozu na palubní desce, virtuální analogové ručičky, tachometr a podobně. Nabídka produktů je už tak rozsáhlá, že je mnohem rychlejší kontaktovat výrobce *Renesas* a nechat si doporučit konkrétní MCU. V nabídce jsou velmi silné všestranné mikroprocesory, například *IMGRH850/C1M-A2* nebo *RH850/D1M*, na trhu již od roku 2015, s velkým počtem pinů, použitelné téměř k čemukoliv. Nicméně i mikroprocesory s přímým účelem, jako je například řízení elektrických či hybridních mo-



Obrázek 1.5: Architektura mikroprocesorů *PIC* [17]

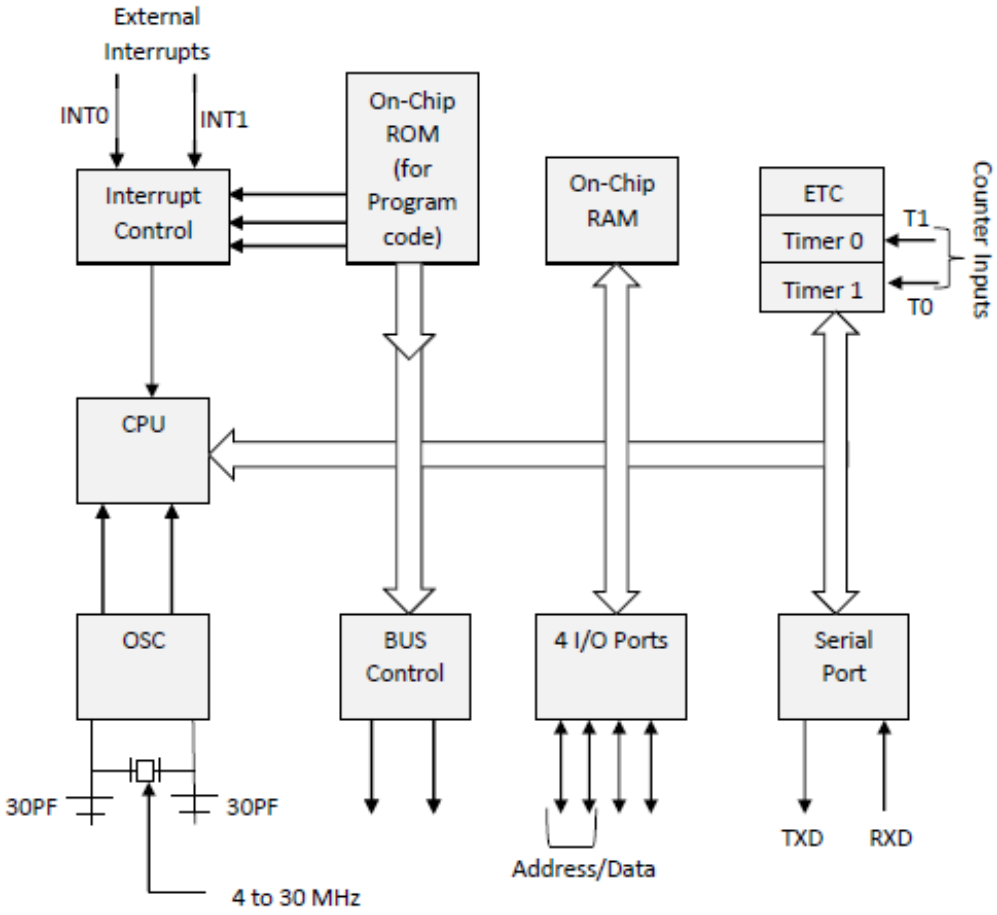
torů, jednoduché zobrazovací panely s měřidly a s 2D vykreslováním, zastoupení pohonné jednotky, kontrola motoru, správa a kontrola podvozku, zastoupení jednotky posilovače řízení a další. [8, 18]



Obrázek 1.6: Pouzdra nejvýkonnějších *Renesas* MCU série *RH850* [19]

1.2.5 Mikrokontrolér *8051*

Mikrokontrolér *8051* vychází z architektury téhož označení, která byla roku 1981 představena společností *Intel*. Veškeré periferie mikrokontroléru jsou připojeny na jednotnou sběrnici (obrázek 1.7), která vede do procesoru. Architektura je tak unikátní a neobvyklá. MCU nabízí 40ti pinový model, z nichž 32 I/O konektorů je plně volitelných. Některé společnosti nabízejí i méně populární verzi s 20 piny nebo i s více piny, nicméně se jedná pouze o zdvojení například napájecích konektorů či jiných periferií. MCU disponuje 4 kB *ROM* pamětí, 128 B *RAM* pamětí a dvěma 16ti bitovými časovači. Architektura obsahuje čtyři paralelní 8mi bitové porty. Výkon 8 bitového procesoru při frekvenci 12 MHz dosahuje hodnoty 10 *MIPS*, výkonnější mohou dosahovat výkonu až 20 *MIPS*. Komunikační možnosti jsou skrze *UART* a *SPI*. Předností tohoto jednoduchého MCU je možnost programovaní v nízko-úrovňovém jazyce, jako je například *assembler*, neboť sada instrukcí je jednoduchá a nekomplexní, údržba a řešení problému jsou rovněž velice jednoduché. Pouzdro je velice kompaktní, flexibilní a MCU se samo nabízí k vykonávání veškerých jednoduchých úloh. MCU samotné je velice levné a poskytuje velkou rychlost vykonávání operací. Stává se tedy častou volbou namísto mikroprocesoru s obecným účelem. *8051* MCU má však i své nevýhody, například nemožnost provádět více operací najednou, jeho struktura je komplexnější oproti mikroprocesoru a složitější úlohy vyžadující paralelismus jsou pro tento typ MCU nerealizovatelné. Příklady aplikací jsou: dálkový ovladač na bázi hesla pro otevření dveří, automatické nouzové osvětlení, řízení světelných signálů dopravy s možností vzdáleného ovládání, spínaní zdrojů, placené parkování na bázi *RFID*, hlasově ovládané robotické vozidlo, přepěťový či podpěťový chránič, domácí automatizace na bázi *Arduino* projektů a další. [8, 20, 21]



Obrázek 1.7: Architektura *8051* [22]

Kapitola 2

Kamery pro automobilový průmysl

Kamery určené pro automobilový průmysl mohou nabývat mnoho účelů od jednoduchých, jako je asistent při parkování, po komplexnější, jako je například sledování okolí a autonomie vozidla. Většina automobilek se snaží kupovat hotová řešení nežli investovat do vlastního vývoje, a tudíž se na trhu vyskytují již kompletní celky obsahující výpočetní výkon, vlastní úložiště pro uložení záznamu, komunikaci pomocí rozhraní *CAN* a nebo další senzorickou část, jako je radar, detektor deště, mlhy či osvětlení neboli lidar a další. Je tak dáno zejména kvůli zodpovědnosti v případě selhání, která padá na hlavu dodavatele. Dnes používané a nabízené technologie umožňují poskytnout panoramatické video přední nebo zadní části vozu, sloužící při parkování nebo při nepřehlednosti povrchu cesty skrze přední část auta nebo komplexnější systémy určené pro autonomii, jako je asistent řidiče. Kamery jakožto elektrické zařízení musí splňovat obdobně jako MCU základní normu *CE*. V následující kapitole budou popsány dostupné kamerové systémy a jejich současné využití. [23, 24, 25]

2.1 Dostupné kamerové systémy

Pod pojmem kamerový systém se nachází komplexní zařízení připravené k použití, obvykle s jedním konektorem, obsahující všechny potřebné výstupy kterými je napájení, uzemnění a komunikace. Existují však i možnosti o vícero konektorech, kde konektor bývá oddělen na napájecí konektor a komunikační konektor. Hojně používaným přenosovým médiem je *Ethernet* a *CAN*. Možnost přenášet informace však lze i skrze I^2C, *LVTD* nebo *CSI-2*. [23, 25]

2.1.1 Náhrada prvků vozidla

Hojně využívanou náhradou prvků vozidla jsou zpětná zrcátka. Kamery jsou umístěny na bočních stranách karosérie v obdobné výši jako by byla umístěná zrcátka, avšak lokace může být jakákoliv. Tohoto je zejména využíváno u super-sportovních vozidel za účelem snížení odporu vzduchu. Obrovskou výhodou je možnost širokoúhlého záběru a tedy pokrytí i úhlů, které by za pomocí

zpětných zrcátek zachytitelné nebyly. Další náhradou prvků vozidla můžou být i skla vozidla, kdy kamery zajišťují vizi kolem vozu a prezentují obraz na palubní systém či obrazovky v interiéru. Tato funkcionalita je hojně využívána v obrněných vozidlech armád. Existují však i prototypy osobních vozidel.

2.1.2 Okolní vize a autonomie

Komplexnější systémy ke svému zpracování vyžadují nejen vizi z mnoha úhlů, ale i informace z dalších senzorických částí. K vyhodnocení všech dat je zapotřebí výkoný hardware, který musí plnit *ASIL* normy. V případě autonomie, kdy je na systém spoléháno a přebírá kontrolu nad vozidlem, pak musí systém garantovat bezporuchovost a schopnost výpočtů v reálném čase. K tomuto patří i řešení různých závad, jako je například nezajištění výpočtů v daném časovém okně, poté přejití k rizikovým procesům, které můžou představovat například zpomalení či zastavení vozidla a podobně. Dodavateli takovýchto systému jsou například *Continental*, *ST* a *ADAS*. Dostupnými systémy, které podporují funkci autonomního řízení od společnosti *Continental* jsou:

- Mono kamera.
- Stereo kamera.
- Multi-funkční kamera s lidarem.
- SVS.

Kamerový systém s jednou kamerou dokáže poskytnout snímek o horizontálním úhlu až 125° a vertikálním úhlu 60° s vysokým rozlišením a možností noční vize. Výpočetní technika je schopná detekovat vozidla, chodce, kola, okraje vozovky, jízdní pruhy a jejich typy. Systém disponuje databází cest, která je aktualizovaná pomocí mobilních dat. Pracovní teploty systému jsou stanoveny od −40 °C až po 95 °C. Zařízení odebírá méně než 7 W při napětí 12 V. Modul nabízí kontrolu světlometů na základě protijedoucích aut, ochranu před vychýlením z jízdního pruhu, asistenta pro udržení v jízdním pruhu, nouzový brzdový systém, adaptivní tempomat včetně detekce dopravního značení a omezení, rozpoznávání semaforů, rozpoznání únavy řidiče a poskytnutí map a databáze silnic. [26]

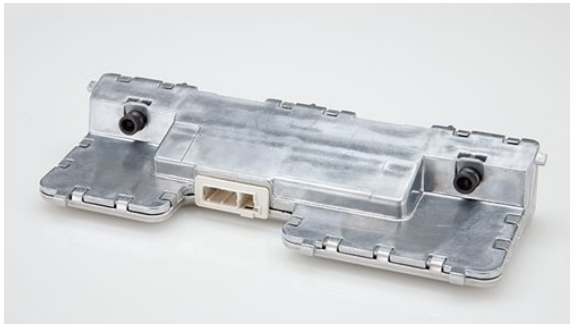
Kamerový systém se dvěma kamerami je schopen detekovat navíc oproti modulu s jednou kamerou překážky na cestě, zvěř, praskliny v cestě, a je schopen určit velikost a vzdálenost k těmto objektům. Nabízené funkcionality jsou shodné s kamerovým modulem o jedné kameře. Tyto kamerové moduly umožňují podporu autonomního řízení.[26, 27]

Kamerový systém multi-funkční kamery s lidarem s označením *MFL4x0* se liší, oproti systémům s jednou či dvěma výkonými kamerami, horší kamerou s maximálním rozlišením 840×630 pixelů o maximálním horizontálním úhlů 40° a vertikálním 29°. Pracovní teploty jsou v rozsahu −40 °C až po 85 °C. Zorný úhel lidaru je 27° pro horizontální úhel a 12° pro vertikální úhel. Lidar je schopen detekovat až na vzdálenost 20 m od vozidla na vlnové délce 905 nm. Výrobce uvádí, že systém garantuje zamezení srážky vozidel při rychlosti do 50 km/h. Pro případ vyšších rychlostí pak systém zpomalí na co nejmenší rychlost a tak zamezí větším škodám a zraněním. Kamerový systém je

schopen detekce objektů, nouzového brzdění, varovat před kolizí, varovat před vybočením z jízdního pruhu, ovládat světlomety a informovat o dopravním značení. [28]

SVS neboli „Surround view system" nabízí naprostý přehled o dění kolem automobilu, kromě prohlížení obrazu přímou cestou lze použít i zobrazení 3D pomocí simulace ve virtuálním světě, a tak poskytnout možnost se rozhlížet kolem automobilu zevnitř automobilu. Výhodou *SVS* systému je možnost autonomního řízení, neboť tento systém má v případě správného vyhodnocování naprostý přehled o dění kolem sebe. Pro případ nepříznivých podmínek, jako je například mlha nebo hustý déšť, lze použít termální kamery. [24, 29]

(a) Mono kamera [30]



(b) Stereo kamera [31]

(c) Multi-funkční kamera s lidarem [32]

Obrázek 2.1: Kamerové moduly společnosti Continental

Kapitola 3

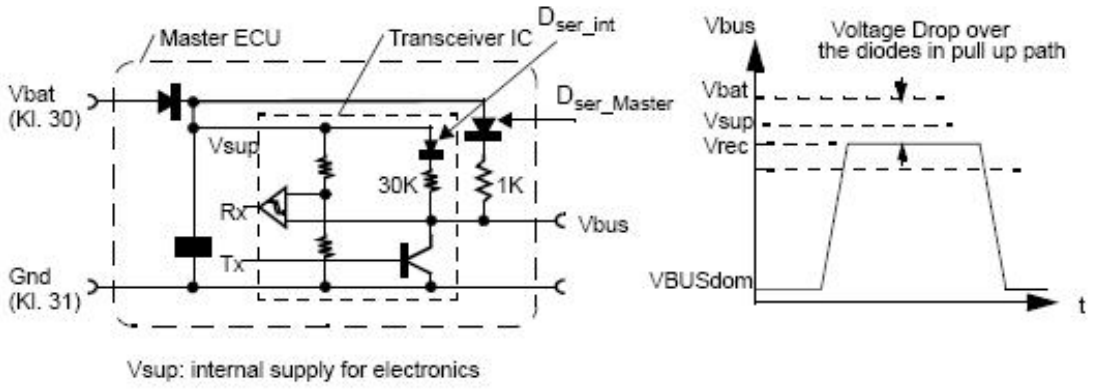
Komunikační možnosti v automobilovém průmyslu

Přenášené informace skrze komunikační média ve vozidle nesou různorodou důležitost. Některá se ztratit mohou bez následků, jiná musí být doručena s určitou garancí, případně musí být opakován pokus o doručení, a některé informace mohou být natolik důležité, že je jejich stav aktualizován každých 5 ms, i když je stav neměnný. Jedná se zejména o světlomety, řídicí části motoru a bezpečnostní systémy. Na sběrnice působí různá rušení, proti kterým je zapotřebí být imunní, nebo alespoň brát tuto chybu v potaz. Může se jednat o vnitřní rušení, kdy rušení vzniká z jiného zařízení ve vozidle, nebo rušení z okolí vozidla. Veškerá elektronika podstupuje různé *EMC* testy a je ověřeno chování systému za kritických stavů. V následující kapitole budou popsány jednotlivé používané sběrnice v automobilovém průmyslu.

3.1 Komunikační rozhraní *LIN*

Komunikační rozhraní *LIN* ve vozidle lze realizovat pomocí jediného vodiče a připojením uzemnění například ke karoserii vozu, což je hlavní výhoda tohoto rozhraní a vhodná zejména pro přenos méně důležitých informací malého počtu a s malou časovou tolerancí. Sběrnice *LIN* zvládá přenést až 2,4 kB/s při maximální délce 40 m a plní koncept OSI modelu. Důvodem vytvoření této sběrnice bylo vytvoření levnější alternativy k sběrnici *CAN*. Příklady použití jsou například stahování okének, elektronické seřizování zrcátek, řízení stěračů a senzor deště. [33]

Přijímač komunikační sběrnice *LIN* pracuje v obousměrném módu. Napájení soustavy je doporučené volit mezi 7 až 18 V. Platí zde pravidlo určování logické úrovně vyjádřené procentuálně, kdy hodnota napětí pod 40% odpovídá logické úrovni 0, a hodnota napětí nad 60 % logické úrovni 1. Touto metodou vzniká dostatečný rozdíl 20 % napájecí hodnoty pro rozeznání 0 a 1. Sběrnice je typu *master–slave* o možnosti připojení vícero *slave* zařízení. [33]



Obrázek 3.1: Vnitřní zapojení přijímače *LIN* [34]

Konfigurace sběrnice je definovaná pomocí souboru *LDF*, který obsahuje informace o rámečcích a signálech odesílané zprávy. Soubor je poté použit v softwarech na všech připojených zařízeních. Zvolené zařízení *master* má na starost odesílání požadavků, kdy je mezi každou zprávou aplikovaná časová mezera, definována souborem *LDF*, která slouží pro příjem dat. Rámeček zprávy (Obrázek 3.2) – neboli rámeček bez pravidel – obsahuje pauzu, synchronizaci a *ID* příjemce. Následuje odmlčení *mastera* a příjem dat. Konec odesílání dat obsahuje kontrolní součet. Komunikaci lze realizovat ve třech módech:

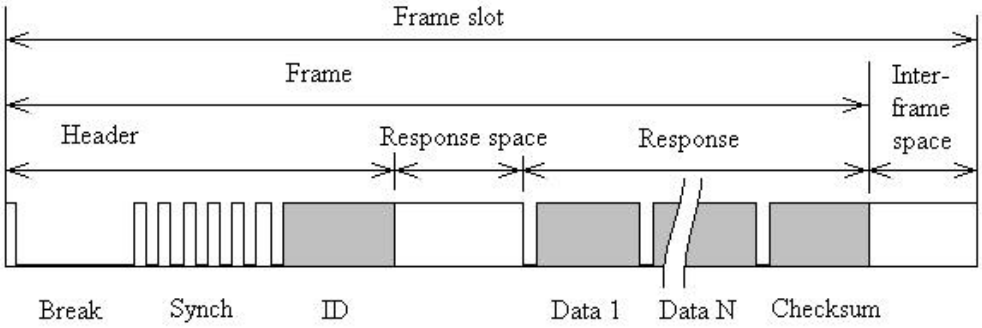
- Rámeček bez pravidel
- Spuštění rámečku při události
- Ojedinělý rámeček

Pro případ spuštění rámečku při události je *master* zařízení v módu odposlechu sběrnice, nezatěžuje ji tedy dotazy a odesíláním rámečků. Událostí se chápe například kolize dvou *slave* zařízení. V takovém případě *master* přejde k odesílání jednotlivých dotazů na zařízení, která způsobila kolizi. Poté přejde opět do stavu odposlechu. Ojedinělý rámeček je odesílán v případě, kdy má *master* zařízení podezření, že došlo k obnově dat. V takovém případě zasílá požadavek na obnovu dat pomocí rámečku. Komunikace probíhá pomocí jednotlivých bajtů. Každý bajt má kromě 8 bitů navíc startovací bit a konečný bit. [33]

Od verze protokolu *LIN* 2.0 je k dispozici možnost diagnostiky sběrnice. Ta je vyvolána odesláním požadavku o 8 bajtech. Odpověď nabývá hodnot 1 až 127 obsahující obecné chyby či stavy, hodnoty od 128 až 255 jsou pak definovatelné stavy programátorem, případně uživatelem. [33]

3.2 Komunikační rozhraní *CAN*

Komunikační rozhraní *CAN* bylo vyvinuto zejména k propojení vzrůstajícího počtu *ECU* a *MCU* ve vozidlech bez nutnosti přítomnosti *master* zařízení. Vzniklý protokol odpovídá standardu *ISO*



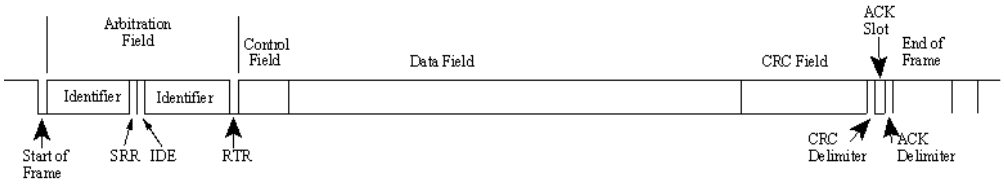
Obrázek 3.2: Rámeček zprávy *LIN* [35]

11898 pro sériovou datovou komunikaci. Sběrnice je realizovaná kroucenou dvoulinkou a nabízí rychlost až 125 kB/s při maximální délce 40 m. Sběrnici lze vést i na delší vzdálenosti, avšak je zapotřebí počítat s menšími rychlostmi. Vzniklý protokol umožňuje snadné použití této sběrnice a v případě návrhu distribuovaného řídicího systému splňuje všechna kritéria ruku v ruce s jednoduchostí. Nabízí velmi dobré zvládnutí chyb na sběrnici v případě komunikačního šumu, rychlé odstranění a opětovné navázání spojení. Rozhraní *CAN* lze implementovat i do nejmenších zařízení na úrovni křemíku, což otevírá možnost pro detekování chyb a řešení selhání. Protokol si poradí i s případným špatně fungujícím uzlem, jeho vyřazením, a tak nedojde k blokování sběrnice. Dnes je původní *CAN* rozhraní nahrazováno rychlejším *CAN FD* s možností flexibilní datové rychlosti zvyšující přenos informací až 10ti násobně. Alternativní cestou je i zvolení komunikačního média *Ethernet*. [36, 37, 38]

CAN rozhraní je typem vysílací sběrnicí, což znamená, že všechna připojená zařízení poslouchají všechny zprávy. Existují však i zařízení tvořící uzly, které odfiltrovávají zprávy a propouští pouze některé. Zprávy samotné jsou krátkého rozsahu, maximálně 94 bitů s odesíláním vždy konkrétnímu zařízení. Protokol umožňuje využít 4 různé zprávy. [36, 37, 38]

- Datový rámeček
- Dotazující se rámeček
- Chybový rámeček
- Rámeček o přetížení

Datový rámeček je nejběžnější formou komunikace. Rámeček je tvořený identifikátorem, daty, *CRC* neboli kontrolním součtem a potvrzovacím polem, které je vyplňováno zařízeními v případě, že byly schopny zprávu přečíst. Pro opačný případ je zpráva opakována. Identifikátor určuje příjemce zprávy. V rozšířené variantě protokolu *CAN 2.0B* je identifikátor tvořen 29ti bitovou hodnotou, pro standardní verzi *CAN 2.0A* se jedná pouze o 11ti bitovou hodnotu. Datová oblast může obsahovat 0 až 8 bajtů dat. Kontrolní součet je vygenerován na základě dat a obsahuje 15ti bitovou hodnotu k rekalkulaci. [36, 37, 38]



Obrázek 3.3: Rozšířený rámeček zprávy *CAN* [39]

Dotazující se rámeček slouží ke vzdálenému požadavku na příjem dat, kdy data nejsou cyklicky posílána za cílem šetření sběrnice. Oproti datovému rámečku obsahuje označení *RTR* bitem v identifikátoru. Princip dotazu funguje tak, že dotazující odešle identifikátor o určité hodnotě. Cílené zařízení odpovídající na zprávu odešle totožný rámeček zpět se stejným obsahem, ale rozšířený o data. Z tohoto vyplývá, že rámeček od odesílatele musí být už připravený pro data a tedy mít stejnou velikost. Pokud tato podmínka nebude splněna, odpověď nebude odeslána. [36, 37, 38]

Chybový rámeček je označení pro rámeček porušující formát rámečku. Je odesílán ve chvíli, kdy je na uzlu zjištěna chyba a dojde k rozšíření této informace o chybě po celé sběrnici do všech uzlů. Přijímač, který odeslal takto chybnou zprávu, se poté pokusí zprávu opakovat. Pro případ neopravení chyby je zde ošetření, aby nedošlo k zahlcení sběrnice chybovými rámečky. [36, 37, 38]

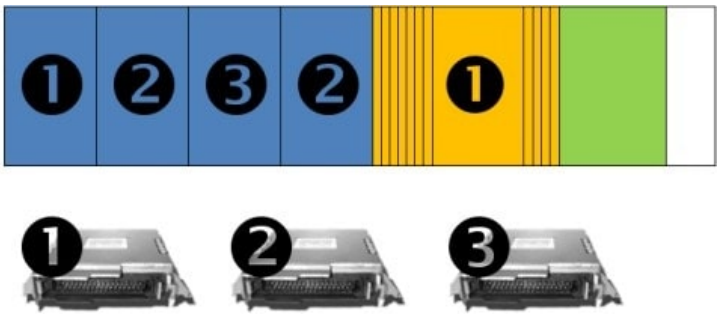
Rámeček o přetížení odesílá zprávu informující o sobě, že aktuálně nedokáže vyhodnotit informace včas. Jedná se tedy o jakousi prosbu, aby byl dotaz opakován později. Není však moc využíván. [36, 37, 38]

3.3 Komunikační rozhraní *FlexRay*

Komunikační rozhraní *FlexRay* dosahuje až 10krát vyšších rychlostí než *CAN* a nabízí vysokou odolnost proti elektromagnetickému rušení. To však přináší i vyšší pořizovací náklady. *FlexRay* rozhraní nabízí možnost komunikace na jednom nebo dvou kanálech současně, kde každý kanál je zastoupen kroucenou dvoulinkou. Topologie sběrnice může být ve svaru hvězda, strom a nebo hybridní. Samotný protokol je typu časově-spouštěný a nabízí možnost deterministického odesílání dat s předpokladem odeslání v rámci mikrosekund. Zatímco uzlům rozhraní *CAN* stačí znát frekvenci komunikace, síť *FlexRay* potřebuje znát i konfiguraci jednotlivých částí sítě, aby mohly mezi sebou komunikovat. Podstatnou výhodou je způsob odesílání zpráv, kdy je využito schéma *TDMA*, tedy rozdělení každého zařízení do časových oken a přiřazení jejich odesílání v těchto časech. Nedochází tedy ke kolizím a lze garantovat determinismus dat. Síťová konfigurace jednotlivých zařízení je pevně definovaná a nelze ji měnit. [40]

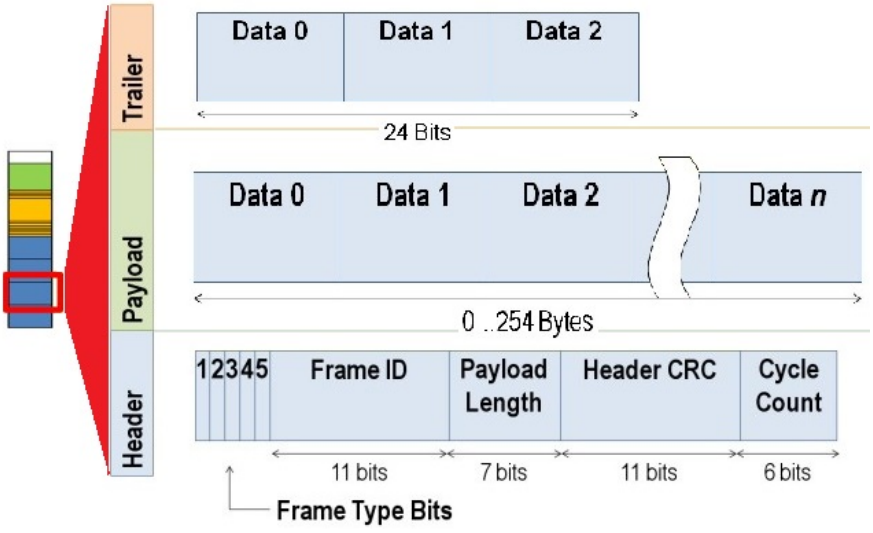
Odesílání dat je tvořeno cyklem, který definuje každému zařízení jeho časové okno kdy může zapisovat. Cyklus je tvořen statickým segmentem, dynamickým segmentem, symbolickým oknem a prodlevou. Statický segment obsahuje časové oblasti (vyobrazeno modře na obrázku 3.4). Každá oblast přísluší definovanému zařízení. Zařízení v tento čas má oprávnění odesílat data. V případě,

že bude nějaké zařízení vypnuto, čas je stále přiřazen a nedojde ke konfliktu. Statické segmenty jsou využívány pro zařízení, u kterých je očekáván pravidelný přenos velkého množství dat. [40] Protože



Obrázek 3.4: Statický segment protokolu *FlexRay* [41]

se v síti mohou nacházet i zařízení, která neodesílají objemné balíčky dat a nebo v nepravidelných intervalech, protokol *FlexRay* disponuje navíc dynamickým segmentem, který má omezenou velikost. Je tedy zapotřebí alokovat velikost dynamické oblasti pro daná zařízení a v případě, že nedojde k obsluze všech zařízení, obměňovat prioritu následujících cyklů. Délka dynamického místa je obvykle jedna mikrosekunda a nazývá se „macrotick". Dynamické schéma je funkcionálně obdobné protokolu rozhraní *CAN*. Symbolické okno je převážně využíváno k údržbě a identifikaci speciálního cyklu, jako je například studený start. Většina vysokoúrovňových aplikací neprovádí interakci s tímto segmentem. Prodleva mezi každým cyklem slouží pro zařízení k přípravě dalších zpráv a případné okamžité odpovědi. [40]

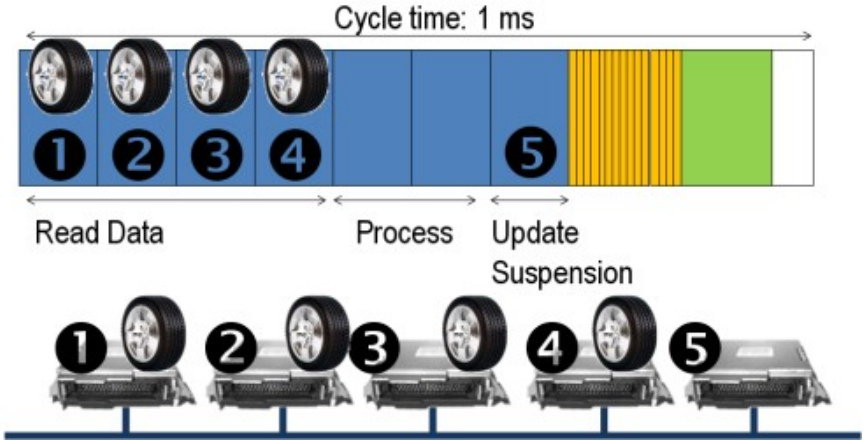


Obrázek 3.5: Obsah rámečku statického segmentu *FlexRay*

Obsah rámečku statického segmentu je znázorněn na obrázku 3.5 a je složen z hlavičky, dat a přívěsu (přeloženo z angličtiny „header, payload and trailer"). Hlavička je tvořena celkem 40ti

bity. Prvních 5 bitů slouží pro určení statusu, dalších 11 bitů pro určení identifikátoru, 7 bitů k určení velikosti dat, následujících 11 bitů pro kontrolní součet hlavičky a posledních 6 bitů jako čítač cyklů. Za hlavičkou následují data, která mohou dosahovat velikosti až 254 bajtů, což je přibližně 30krát více, než umožňuje rozhraní *CAN*. Přívěs obsahuje kontrolní součet pro data. [40]

Příklad aplikace protokolu *FlexRay* lze ukázat na adaptivním odpružení automobilu (obrázek 3.6), kdy v prvních čtyřech statických oblastech probíhá sběr informací o náklonu kol, rotaci a poloze kol vůči karoserii. Poté je aplikováno časové okno pro vyhodnocení těchto informací další řídící jednotkou a ještě ve stejném cyklu jsou okamžitě aplikovány změny podvozku. [42]



Obrázek 3.6: Příklad použití *FlexRay* protokolu [42]

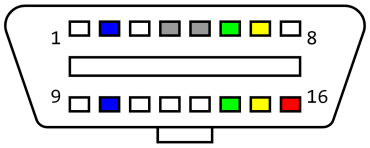
3.4 Komunikační rozhraní *OBD*

Jak již název „On-Board-Diagnostics" napovídá, jedná se o rozhraní sloužící primárně k diagnostice vozidla servisními techniky. Z počátku se diagnostika využívala pouze pro kontrolu správného fungování emisí (*ALDL* a *OBD-I*), nicméně s modernizováním vozů a velkého množství elektroniky se stala diagnostika nezbytnou součástí každého vozu. Vznikl standard *OBD-II*. Podmínkou standardu bylo umístění konektoru v interiéru a to v okolí 0,61 m od řidiče. Další možné využití lze najít pro vylepšení výkonu řídicí jednotky, shromažďovaní dat v čase, zjišťování závad hluboko v systému nebo všeobecnou analýzu. [43]

Standard *OBD-II* definuje standardizovaný 16ti pinový konektor a komunikační protokoly. Ke každému pinu konektoru náleží specifikované funkce a komunikační protokoly. Každá automobilka využívá svých specifických protokolů a přiřazení funkcí na volné piny. V rámci normalizace se nesmí přenášet funkce pinů. Pin 2 reprezentuje kladný vodič protokol *SAE J1850* značený na obrázku 3.7 modrou barvou. Pin číslo 4 slouží jako uzemnění ke karosérii a pin 5 uzemnění signálových vodičů. Pracovní úroveň napětí je 5 V. Délka zpráv je pevně stanovena na 12 bajtů a obsahuje kontrolní součet. Pro protokol komunikující pomocí rozhraní *CAN*, tedy *ISO 15765-4* a *SAE J2284*, jsou

vyhrazeny piny 6 a 14, značeny zelenou barvou, kde pin 6 značí pozitivní vodič. Pro protokoly *ISO 9141-2* and *ISO 14230-4* – známe také jako *K-line* – jsou použity piny 7 a 15, kde se opět nachází pozitivní část v horní části konektoru, tedy na pinu 7. Délka zpráv je stanovená maximálně na 260 bajtů. Zprávy samotné mají charakter *UART*. Pin 16 představuje napětí z baterie. Ostatní nejmenované piny, jsou přiřazeny a rozděleny výrobcem a mohou sloužit speciálním účelům, například jako *Ethernet*, *RPM* signál, status signál, spínání zapalování a další. [43]

Standard *EOBD* vznikl na základně Evropského zákona a jedná se o obdobu standardu *OBD-II*. Byly například sjednoceny kódy selhání pro všechny osobní automobily. To vše již roku 2001. [43]



Obrázek 3.7: Standardizovaný konektor *OBD-II* [44]

3.5 Komunikační rozhraní *MOST*

Komunikační rozhraní *MOST* je multimediální sběrnice využívána zejména k přenosu velkého objemu dat, jako je například i audio a video přenos. Rozhraní definuje všech 7 vrstev *ISO* modelu. Podle použitého protokolu lze volit mezi propojením optickým vláknem, skrze *UTP* nebo koaxiálním vodičem. Číslo protokolu definuje i rychlost sběrnice v Mb/s, jedná se o protokoly *MOST25*, *MOST50* a *MOST150*. Nejčastěji je voleno optické propojení díky nezávislosti vůči elektromagnetickému jevu. Sběrnice podporuje funkci „plug and play". Komunikační rozhraní *MOST* je obdoba ethernetové sběrnice. Častá topologie sítě bývá hvězdicová nebo kruhová. Použití rozbočovačů je také možné, nicméně při použití v automobilu nepraktické a tedy nepoužívané. [45]

Kapitola 4

Možnosti zpracování syntézy a zpracování obrazu

Ke zpracování obrazu lze využít nemalého množství knihoven nabízející rozdílná řešení, správu a hardwarového využití. Knihovny jsou mnohdy pouze pro konkrétní programovací jazyk a jen malé množství je modulární. Pro programovací jazyky *c++*, *c#* a *python* lze využít například knihovny *SIGIL* nebo *DLIB*. Pro programovací jazyk *java* například *ImageJ*, *LeadTools* nebo *JavaCV* sestávající se z několika populárních knihoven, jako je *OpenCV*, *FFmpeg*, *OpenKinect*, *videoInput* a dalších. Modulárními knihovnami jsou například *OpenCV*, kterou lze využít pro všechny již jmenované jazyky a nebo *Magick* pro jazyk *c++* a *java*. Knihovna *OpenCV* je však ze zmiňovaných nejvyužívanější díky rozsáhlým možnostem funkcí, podporou pro různé operační systémy a jazyky. V následující kapitole bude popsaná knihovna *OpenCV* a vysvětlena problematika spojování snímků. [46, 47]

Knihovna *OpenCV* umožňuje aplikaci na takřka všech platformách. Podporu lze najít pro *Windows*, *Linux*, *iOS*, *Android* a platformy typu *ARM*. Podmínkou použití je přítomnost operačního systému, neboť samotné knihovny využívají systémových knihoven nezbytných pro správné fungování. Knihovna *OpenCV* je dodávaná v surové podobě zdrojových kódů, které jsou následně pomocí nástroje *cmake* konfigurovány a připraveny k sestavení. Konfiguruje se například druh kompilátoru, definování typů knihoven, definování cest k systémovým knihovnám a cest k dodatečným souborům instalace. Po vygenerování jsou kontrolovány chyby konfigurace. Nejčastějšími jsou špatně uvedené cesty nebo požadavek využít knihovny, které systém nemůže poskytnout. V případě ignorování těchto chyb buďto selže následné sestavení, nebo bude nastavení ignorováno a k sestavení těchto knihoven nedojde. Nedílnou součástí pro zpracování obrazu disponující knihovny *OpenCV* je podpora pro prací s architekturami *CUDA* a *OpenCL*. Sestavení je doporučeno dělat vždy na každém operačním systému se specifickým hardwarem zvlášť. Knihovny lze však sestavit i pro jiný systém, je však nutné mít k dispozici všechny systémové knihovny daného systému. [46]

4.1 Získání klíčových bodů a deskriptorů snímku

Knihovna *OpenCV* nabízí široké množství prací s obrazem od nejzákladnějších až po složitější algoritmy. Mezi základní operace patří například převod obrazu do odstínu šedi, prahování, segmentování, spojování obrazů, vytváření histogramů nebo vkreslování tvarů. Mezi složitější algoritmy pak patří filtrování, geometrická transformace, smíšená transformace, planární dělení, analýza struktur a tvarů deskriptorů, analýza pohybu, sledování objektů, detekce bodů zájmu a detekce objektů. Pro každé jmenované téma je napsaná separovaná třída s dostupnými metodami. Díky dostatečné dokumentaci a dostupným příkladům je možné snadno vytvořit vlastní algoritmus. Názvy metod jsou mezi jednotlivými programovacími jazyky totožné, pozor si je třeba dát pouze na datové typy a syntaxi. Nepatrný rozdíl lze najít například i v počtu parametrů, kdy například v programovacím jazyce *python* lze požadovat i několik návratových hodnot, tudíž předpis metody obsahuje pouze vstupní parametry. V programovacím jazyce *c++* předpis funkcí často obsahuje i výstupní parametry s referencí, neboť nelze vracet několik návratových hodnot. Datové typy knihovny *OpenCV* jsou programované tak, aby nebylo nutné psát referenční znak. Jako příklad je uveden předpis metody *initUndistortRectifyMap()* ve výpisu 4.1, který je součástí geometrické transformace, pro jazyky *python* a *c++*. [48]

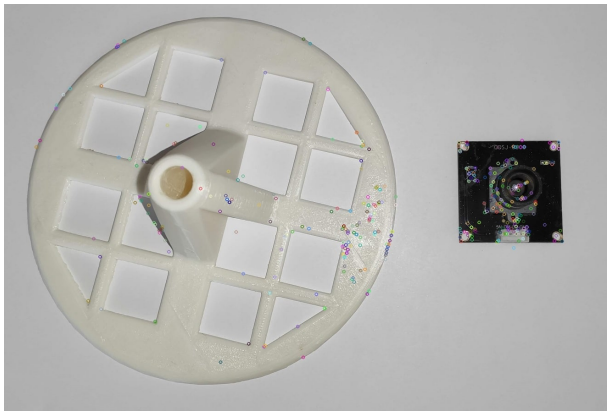
```
// c++
cv::initUndistortRectifyMap (InputArray cameraMatrix, InputArray distCoeffs, InputArray R,
                             InputArray newCameraMatrix, Size size, int m1type,
                             OutputArray map1, OutputArray map2);

// python
map1, map2 = cv.initUndistortRectifyMap ( cameraMatrix, distCoeffs, R, newCameraMatrix, size,
                                          m1type [, map1[, map2]])
```

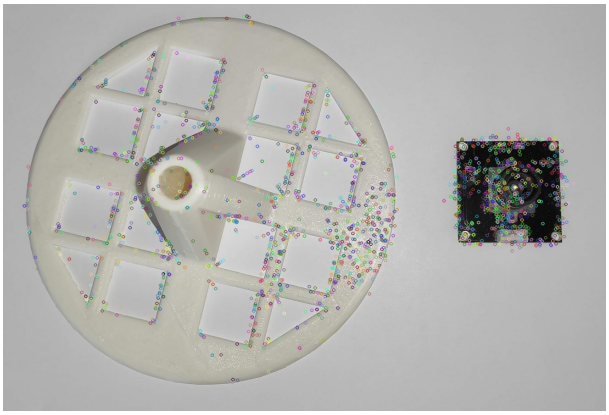
Výpis 4.1: Příklad syntaxe *OpenCV* v různých jazycích [1]

Po detekci klíčových bodů a získání deskriptorů ze dvou snímků lze provést jejich porovnání. Na základě tohoto porovnání následně vyhodnotit vzájemnou homografii, díky které lze provést transformaci perspektivy snímku. Uplatnění lze nalézt například pro vyrovnání fotografie, vytváření panoramatu nebo vyříznutí objektu ze snímku a následné vyrovnání.

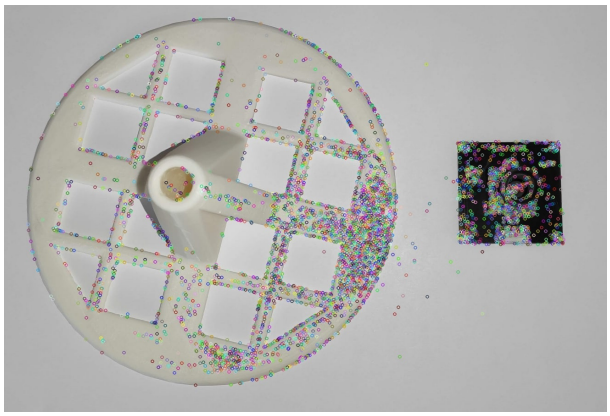
Pro získání klíčových bodů a následně vyjádření deskriptorů lze využít několik algoritmů navzájem specifickými pro svou aplikaci. Tyto detektory vyhledávají v obrazu specifické lokace v ideálních podmínkách univerzální oproti ostatním lokacím. Čím specifičtější informace je, tím přesnější je výstupní informace a tedy i správné vyhodnocení v následujících operacích, tedy porovnávání lokací – neboli deskriptorů – mezi dvěma snímky. Příklady algoritmů obsažených v knihovně *OpenCV* jsou *Harrisova detekce rohů*, *Shi-Tomasova detekce rohů*, *SIFT*, *SURF*, *FAST*, *BRIEF* a *ORB* algoritmy. Každý z těchto algoritmů využívá specifických podmínek za účelem najití klíčových bodů. Na obrázku 4.1 jsou zobrazeny výsledky jednotlivých algoritmů pro detekci klíčových bodů. [49]



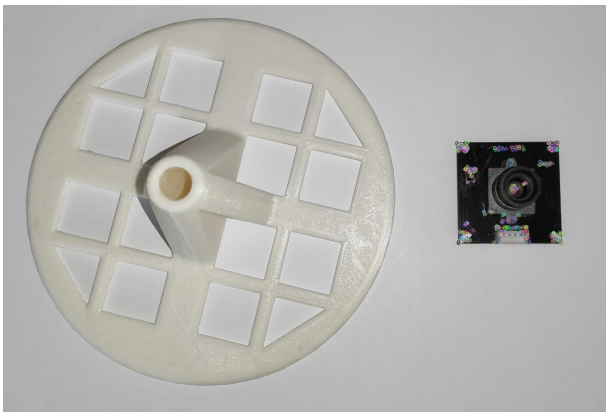
(a) algoritmus *SIFT*



(b) algoritmus *SURF*



(c) algoritmus *FAST*



(d) algoritmus *ORB*

Obrázek 4.1: Vyhodnocení klíčových bodů různými algoritmy

4.1.1 *Harrisova a Shi-Tomasova detekce rohů*

Jeden z prvních pokusů o nalezení rohů provedli *Chris Harris* a *Mike Stephens* v roce 1988. Vzniklá metoda byla pojmenovaná jako *Harrisova detekce rohů*. Jednoduchá myšlenka byla přenesena do matematické podoby. Hledána je intenzita v posunutí *(u, v)* ve všech směrech. Matematické vyjádření je uvedeno v rovnici číslo 4.1. [50]

$$E(u,v) = \sum w(x,y) \cdot [I(x+u, y+v) - I(x,y)]^2 \tag{4.1}$$

Funkce okna je buď *obdélníková*, nebo *Gaussova*, které přiřazují váhu pixelům. Funkci $E(u,v)$ je zapotřebí maximalizovat pro možnost detekce rohů. To znamená, že je zapotřebí maximalizovat druhou mocninu. Použitím *Taylorovy expanze* na výše uvedenou rovnici lze získat rovnici 4.2. Matici $\mathbf{M}$ lze pak rozšířit o rovnici 4.3. [50]

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \cdot \mathbf{M} \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.2}$$

kde

$$\mathbf{M} = \sum_{x,y} w(x,y) \cdot \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{4.3}$$

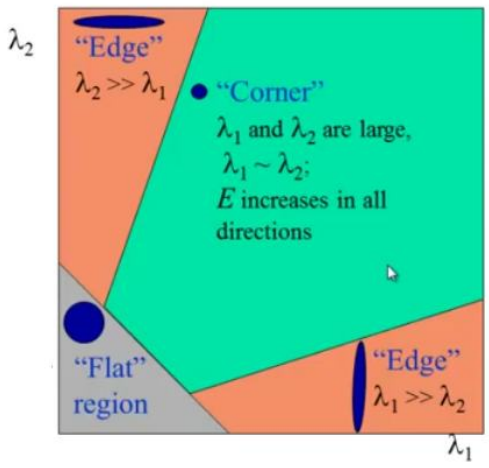
Proměnné I_x a I_y v rovnici 4.3 představují deriváty obrazu ve směrech x a y. Hlavní částí je pak skóre R, které určuje, zda oblast obsahuje roh, či nikoliv. Výpočet skóre R je reprezentováno rovnicí 4.4. [50]

$$R = \det(\mathbf{M}) - k \cdot (\mathrm{trace}(\mathbf{M}))^2 \tag{4.4}$$

Rovnice 4.4 lze upravit do tvaru:

$$R = \lambda_1 \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2 \tag{4.5}$$

Výsledná hodnota R rozhoduje o dané oblasti, zda se v ní nachází roh, hrana a nebo se jedná o plochu. V případě, že je hodnota $|R|$ je nízká, jedná se o plochu. Stává se v případě, kdy jsou λ_1 a λ_2 nízké hodnoty. Pokud je R menší než 0, oblast je hrana. Nastává v případě, kdy je λ_1 mnohem větší než λ_2 nebo naopak. Pokud je hodnota R větší než nula, jedná se o roh. Pro tento případ jsou λ_1 a λ_2 buďto o velmi vysoké hodnotě, nebo skoro rovny. Zmiňované vyhodnocení lze vyjádřit obrázkem 4.2. [50]

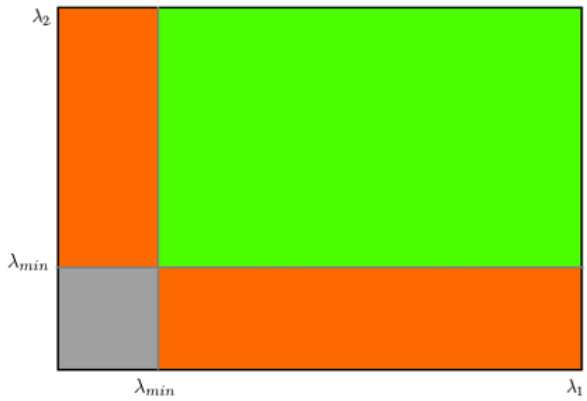


Obrázek 4.2: Grafické znázornění *Harrisovy detekce rohů* [51]

Později roku 1994 *J. Shi* a *C. Tomas* provedli úpravu původní *Harrisovy rovnice*, která vykazuje lepších výsledků. Původní rovnice 4.4 byla upravena do tvaru rovnice 4.6. [50, 52]

$$R = \min(\lambda_1 \lambda_2) \tag{4.6}$$

Pokud je hodnota R větší než prahová hodnota, jedná se o roh. Grafické znázornění odpovídá obrázku 4.3. Z obrázku je vidět, že pouze pro případ, kdy jsou λ_1 a λ_2 větší než minimální hodnota λ_{min}, jedná se o roh. Odpovídá zelené oblasti v obrázku.



Obrázek 4.3: Grafické znázornění *Shi-Tomasovy detekce rohů* [53]

4.1.2 *Scale Invariant Feature Transform – SIFT*

Harrisův a *Shi-Tomashův* algoritmus pro detekci rohů je mocným nástrojem, pokud ale bude snímek škálován – zvětšován, roh, který by byl algoritmy objeven, je ztracen. Roh tentokrát nezabírá pouze

jednu oblast, ale je rozprostřen přes několik oblastí najednou. V roce 2004 *PhD. David Lowe* přišel s novým algoritmem nazvaným *Scale Invariant Feature Transform*, který extrahuje klíčové body a počítá jeho deskriptory. Algoritmus *SIFT* zahrnuje čtyři kroky. [54, 55]

- Detekce extrémního měřítka v prostoru
- Lokalizace klíčových bodů
- Přiřazení orientace
- Deskriptor klíčových bodů

Abychom mohli detekovat roh rozprostřený do několika oblastí, je třeba provést filtrování měřítka a prostoru. Menší prostor je zvětšen a je nalezen *Laplaceův Gaussioův filtr* o různých hodnotách σ. *LoG* funguje jako detektor kružnic, který detekuje objekty kružnicemi o různých velikostech v důsledku v závislosti na hodnotě σ, σ funguje jako parametr měřítka. Vysoká hodnota σ znamená velkou pravděpodobnost najití rohu o velkém rozměru, zatímco nízká hodnota σ znamená najití malého rohu. Lze tedy nalézat lokální maxima v celé škále a prostoru, což poskytuje seznam souřadnic x a y a hodnot σ. Lze říci, že existuje potenciální klíčový bod *v(x, y)* s měřítkem σ. [54, 55]

Takový algoritmus je však velice výpočetně náročný, a tak je využito diference *LoG*. Rozdíl se získá rozostřením obrazu se dvěma různými hodnotami σ. Tento proces se provádí pro různé oktávy obrazu v *Gaussově pyramidě*. Jakmile je diferenční *DoG* nalezen, snímky jsou prohledávány pro nalezení lokálních extrémů v daném měřítku a prostoru. Příkladem hledání je procházení pixelu po pixelu a porovnávání 8 sousedních bodů. Pokud je nalezen lokální extrém, bod je označen jako klíčový bod v daném měřítku. [54, 55]

Po získání potenciálních klíčových bodů je zapotřebí filtrovat jen ty nejrelevantnější. K získání přesnějšího umístění extrémů je využito *Taylorovy série* rozšíření rozsahu prostoru, a pokud je intenzita v tomto extrému menší než prahová hodnota 0,03, jsou hodnoty odstraněny. Obdobný způsob je aplikován i pro hrany, kdy *DoG* reaguje s vyšší odezvou v případě objevení hran. Obdobně jako u *Harrisova detektoru rohů*, kde byly porovnávány dvě hodnoty, i zde jsou hodnoty porovnávané nicméně nikoliv mezi sebou, ale se stanovenou hodnotou prahu, kdy všechny hodnoty menší, než tento práh jsou odstraněny z výběru. Díky této metodě zůstávají pouze klíčové body s vysokou váhou. [54, 55]

Po získání a odfiltrování jen nejvhodnějších klíčových bodů je zapotřebí těmto bodům přiřadit jejich orientaci v prostoru. Toto definování pomáhá při další úpravě snímku v případě rotací, kdy za předpokladu, že známe původní orientaci, můžeme snadno vypočíst orientaci novou. Algoritmus vytváří histogram orientací se 36ti segmenty rozdělenými do SI360° a daný segment přiřazuje ke klíčovému bodu. Vytvoření klíčových bodů se stejným umístěním a měřítkem, ale různými směry, přispívá ke stabilitě párování. V poslední fázi, kdy o vytvořených klíčových bodech je známo

do všech detailů, lze vytvořit deskriptory prezentující klíčové body. Deskriptor je tvořen z okolí klíčového bodu o velikosti 16×16 pixelů a je rozdělen na 16 dílčích bloků o velikosti 4×4. Pro každý blok je vytvořen na základě vstupních informací 8mi hodnotový histogram, celkově je k dispozici 128 hodnot. [54, 55]

4.1.3 *Speeded-Up Robust Features – SURF*

Z algoritmu *SIFT* vychází algoritmus *SURF*, neboli *Speeded-Up Robust Features*, který je o něco rychlejší. Implementuje navíc filtrací *LoG* jehož největší výhodou je, že konvoluce s boxovým filtrem lze snadno vypočítat pomocí integrálních obrazců, což lze provádět paralelně pro různá měřítka. Oproti *SIFT* je upraveno vyhodnocování orientace, která je stanovena na základě sousedních měřítek klíčového bodu. [56]

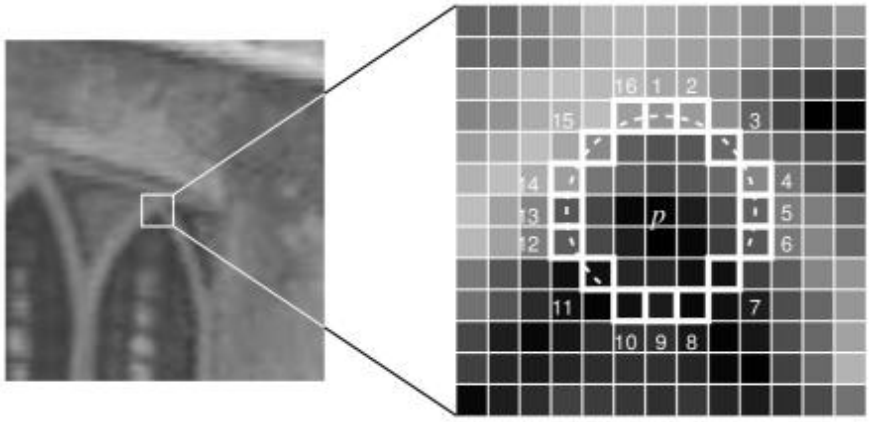
4.1.4 *Binary Robust Independent Elementary Features – BRIEF*

Algoritmus *SIFT* pro svůj výpočet vyžaduje spoustu paměti, pouze pro jeden deskriptor až 512 bajtů. Pro algoritmus *SURF* je zapotřebí až 256 bajtů. Takovýchto potencionálních deskriptorů mohou být i tisíce. Zde se již může narazit na problém s dostupným hardwarem například na embedded platformách nebo kompaktních zařízeních. Pro algoritmus *BRIEF* je prvním krokem převedení hodnot s plovoucí čárkou na binární řetězce, lze použít několika komprimovácích metod jako je například *PCA*, *LDA* nebo *LSH*. Vzdálenosti jsou poté vyhodnocovány za použití XOR funkcí a bitů, které jsou v modernější CPU velmi rychlé. Toto řešení však stále neřeší problém s pamětí, neboť je nejdříve zapotřebí najít deskriptory a až poté porovnávat vzdálenosti. *BRIEF* algoritmus obchází hledání deskriptorů a umožňuje vyhodnotit úrovně na základě vstupních souřadnic. Algoritmus vybírá sousední oblasti bodů a porovnává vzájemné intenzity. Algoritmus je schopen získat pouze deskriptory, neumí detekovat klíčové body. Pro získání bodů lze použít jiných algoritmů. [57]

4.1.5 *Features from Accelerated Segment Test – FAST*

Jak již vyplývá z názvu algoritmu, jedná se o velmi rychlý algoritmus cílený k vyhodnocování v reálném čase. Algoritmus však dokáže vyhodnotit pouze klíčové body. Pro výpočet deskriptorů je zapotřebí použití jiných algoritmů. Princip algoritmu je velice prostý, nejprve je zvolen bod. Na základě jeho hodnoty je určena prahová hodnota. Dále je porovnáno 16 pixelů v kruhovém uskupení od vybraného bodu. Pokud počet okolních bodů, s intenzitou vyšší nebo naopak nižší než zvolený práh, je větší než stanovená hodnota n, pak je zvolený bod označený jako roh a vybrán jako klíčový. Hodnota n procházených bodů je obvykle volena na hodnotu 12 bodů. Než je však detekováno všech n hodnot v okolí, je vykonána takzvaně rychlostní zkouška, která prohledá body vzdálené od sebe 90°. Porovnávají se vždy intenzity protějších bodů. Pokud aspoň tři body jsou shodné, tedy tmavší nebo světlejší, než je stanovený práh, pak je vykonán kruhový algoritmus detekující všechny n hodnoty. Algoritmus je sám o sobě velmi rychlý, avšak má i své slabiny. I pro 12

shodných bodů v okolí není schopen dostatečně eliminovat všechny špatné výběry. výběr výchozích bodů není optimální. Může nastat i situace, kdy u vysokorychlostního testu neprojde roh. Detekování vícero rohů vedle sebe může vést k detekování jednoho totožného rohu několikrát. Na obrázku 4.4 lze vidět průchod algoritmu okolními body. Jako první jsou porovnány body 1 a 9, pokud se rovnají, jsou porovnány body 5 a 13. Jakmile je splněna rychlostní zkouška, přechází se k porovnání bodů v kruhovém uspořádání. [58, 59]



Obrázek 4.4: Grafické znázornění kruhového výběru *FAST* algoritmu [60]

4.1.6 Orientovaný *FAST* a rotovaný *BRIEF* algoritmus – *ORB*

Algoritmus byl vyvinut vývojáři zabývajícími se vývojem *OpenCV* v roce 2011. *ORB* algoritmus je sestaven z upraveného algoritmu *FAST* pro získání klíčových bodů a algoritmu *BRIEF* pro získání deskriptorů rovněž o několika úpravách pro zvýšení výkonu. Jelikož algoritmus *FAST* nevypočítává orientaci v prostoru pro jednotlivé klíčové body, bylo zapotřebí dodatečných výpočtů. Toho bylo docíleno výpočtem těžiště z několika okolních bodů nacházející se v kruhové oblasti o daném poloměru. Vektor daný klíčovým bodem a těžištěm udává směr orientace. Algoritmus *BRIEF* nebere v potaz orientaci klíčových bodů. Proto jsou data po získání deskriptorů upravena pro získání správných rotací. Další zápornou vlastností je velký rozptyl a střední hodnoty obvykle blízké hodnotě 0,5. Pokud je však orientace ve směru klíčových bodů, tato vlastnost se vytrácí a výběr se stává nekonzistentní. Pro stejný vstup je poskytnut různý výsledek. Pro správný chod algoritmu je zapotřebí zajistit nezávislé testy jednotlivých deskriptorů, aby každý test přispěl ke správnému výsledku. K eliminaci těchto jevů je provedena filtrace hodnot pouze o vysokém rozptylu a se střední hodnotou blízko 0,5 a zároveň mezi sebou nekorelující, nazývané jako *rBRIEF*. Algoritmus *ORB* je mnohem rychlejší než *SURF* nebo *SIFT* a výpočet deskriptorů funguje lépe než pro algoritmus *SURF*. Vhodnou aplikací můžou být zařízení s nízkou spotřebou energie. [61]

4.2 Vyhodnocování podobností dvou snímků

Pro vyhodnocení podobností existují v knihovně *OpenCV* dva algoritmy, *BruteForce* a *FLANN*. Oba algoritmy umožňují na základě deskriptorů dvou snímků vyhodnotit shody mezi dvěma obrazci. Algoritmus *FLANN* využívá optimalizované algoritmy pro efektivní hledání nejbližších sousedních deskriptorů a je vhodný zejména pro porovnávání velkého množství dat a dimenzionálních struktur. Algoritmus je rychlejší než *BruteForce* znatelně pro větší počet dat. Princip porovnávání algoritmu *BruteForce* spočívá v selekci jednoho deskriptoru ze snímku prvního a porovnání se všemi deskriptory snímku druhého. Při volbě metod pro výpočet jsou zapotřebí volit dva parametry, které závisí na zvolených algoritmech pro výpočet klíčových bodů a deskriptorů. Při špatně zvolených parametrech nemusí být výsledky správné, případně může výpočet selhat úplně. Například kombinace *ORB* detektoru a *FLANN* porovnávače se základními parametry selže. Knihovna *OpenCV* umožňuje pro oba algoritmy zvolit typ výpočtu, buďto najít pouze nejlepší shodu – zvolená metoda se nazývá *match()*, a nebo najít několik nejlepších shod, zvolaná metoda se nazývá *knnMatch()*. Jak lze vidět z obrázku 4.5, oba algoritmy vyhodnocují rozdílné výsledky. Pro oba algoritmy byl nalezen stejný počet shod a to 347. Lze však vidět jisté odlišnosti, kdy algoritmus *FLANN* tvoří spousty ne-vodorovných spojitostí. Avšak nejedná se přímo o chyby, neboť ku příkladu, nalezené okraje rohů kamer jsou totožné jak v horní části na podstavcích, tak i vystavený kus ve spodní části. Algoritmus tedy správně spojuje totožné rohy, obdobných příkladů v obrázku 4.5 je několik. [62]

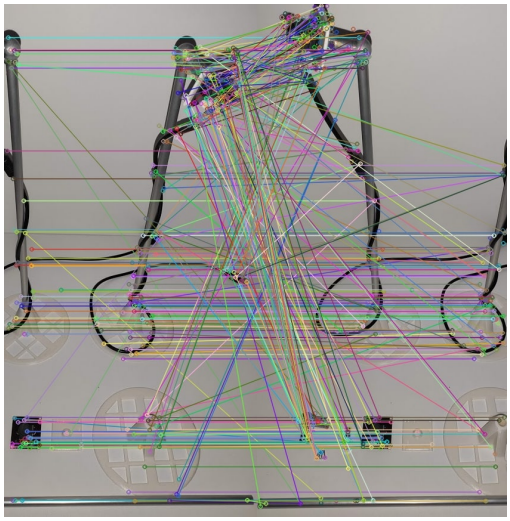
4.2.1 Vytvoření panoramatu

Po získání klíčových bodů ve dvou obrazcích, vyjádření deskriptorů a nalezení shod lze vyjádřit vzájemnou homografii mezi dvěma snímky maticí o velikosti 3×3 reprezentující perspektivní transformaci jednoho ze snímku v prostoru. Předtím, než je však výpočet matice proveden, je zapotřebí vyfiltrovat pouze podobné shody o podobné vzdálenosti. Je tedy často aplikována poměrová filtrace. Tu lze aplikovat dvěma způsoby. Buďto vzdálenost předchozích deskriptorů musí být aspoň 70 % procházených a nebo je zjištěna nejkratší vzdálenost shodných dvou deskriptorů a je povolen pouze k násobek vzdálenosti. Příklad kódu lze vidět ve výpisu 4.2.

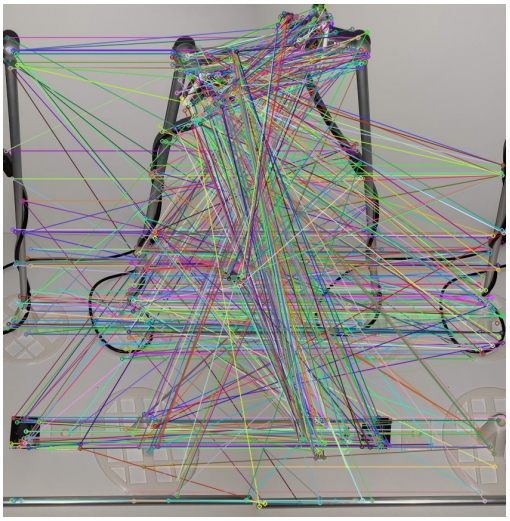
```
// ---------------- Poměrová filtrace ----------------
for (size_t i = 0; i < matches.size(); i++) {
    if (matches[i][0].distance < 0.7f * matches[i][1].distance) {
        goodMatches.push_back(matches[i][0]);
}   }
// ---------------- filtrace - k násobek ----------------
for (int i = 0; i < matches.size(); i++) {
  if (dist < min_dist) min_dist = matches[i][0].distance;
  if (dist > max_dist) max_dist = matches[i][0].distance;
}
for (int i = 0; i < matches.size(); i++)
  if (matches[i][0].distance < 3 * min_dist) goodMatches.push_back(matches[i][0]);
```

(a) Vstupní snímek levá strana

(b) Vstupní snímek pravá strana



(c) algoritmus *BruteForce*

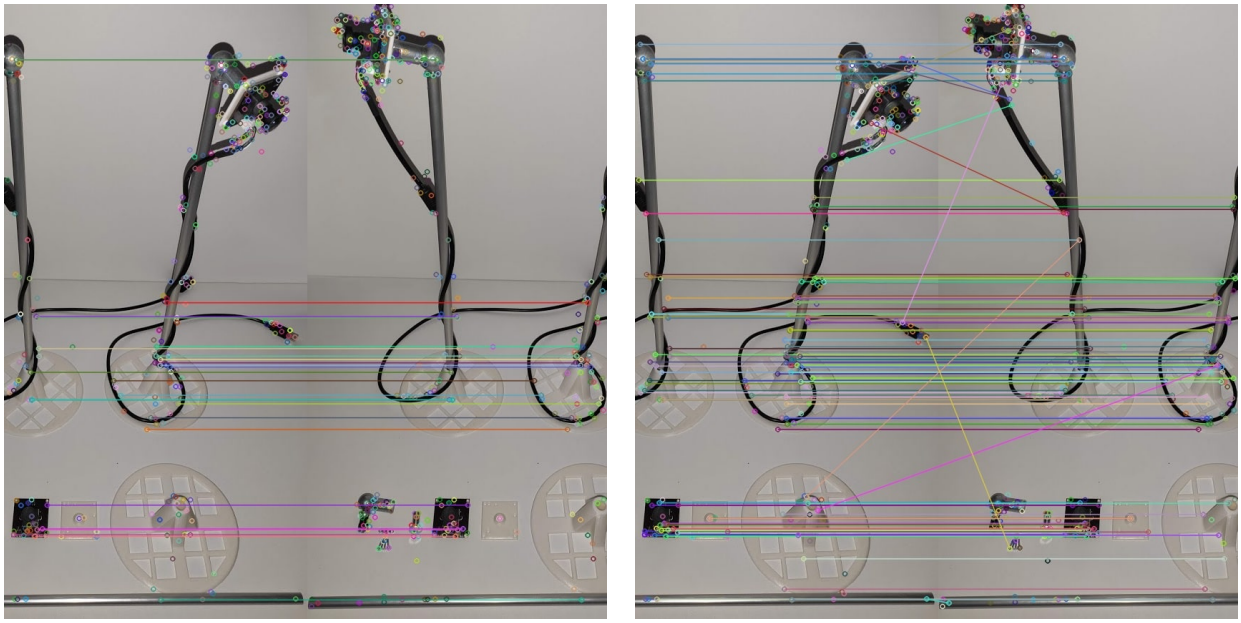


(d) algoritmus *FLANN*

Obrázek 4.5: Vyhodnocení podobností, klíčových bodů a deskriptorů algoritmem *SIFT*

Výpis 4.2: Příklad provedení filtrace shod

Pro ukázku výsledků filtrace byly použity stejné snímky jako na obrázku 4.5. Na obrázku 4.6 si lze všimnout úplného odfiltrování nesouběžných shod pro algoritmus *BruteForce*. Pro algoritmus *FLANN* odfiltrování všech nesouběžných shod nebylo zcela úspěšné, nicméně v malém množství nemají tyto špatné shody vliv při výpočtu homografie. Odfiltrování i těchto shod by vyžadovalo snížení poměrového koeficientu.

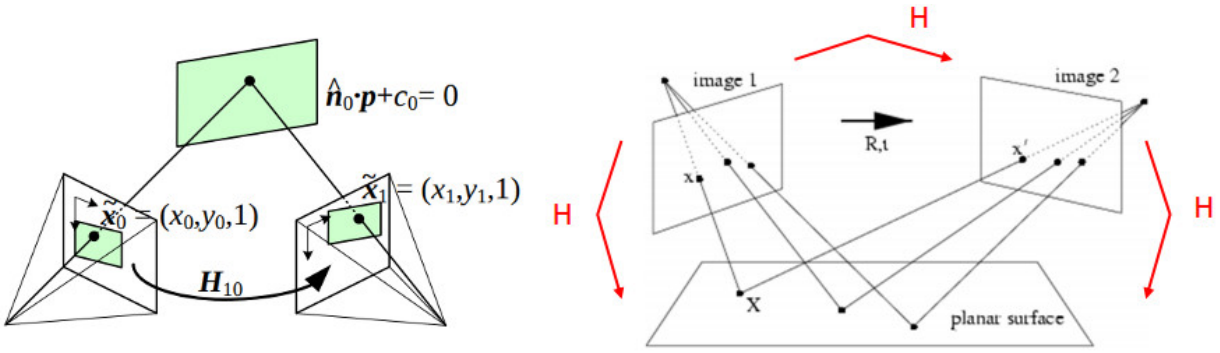


(a) algoritmus *BruteForce*

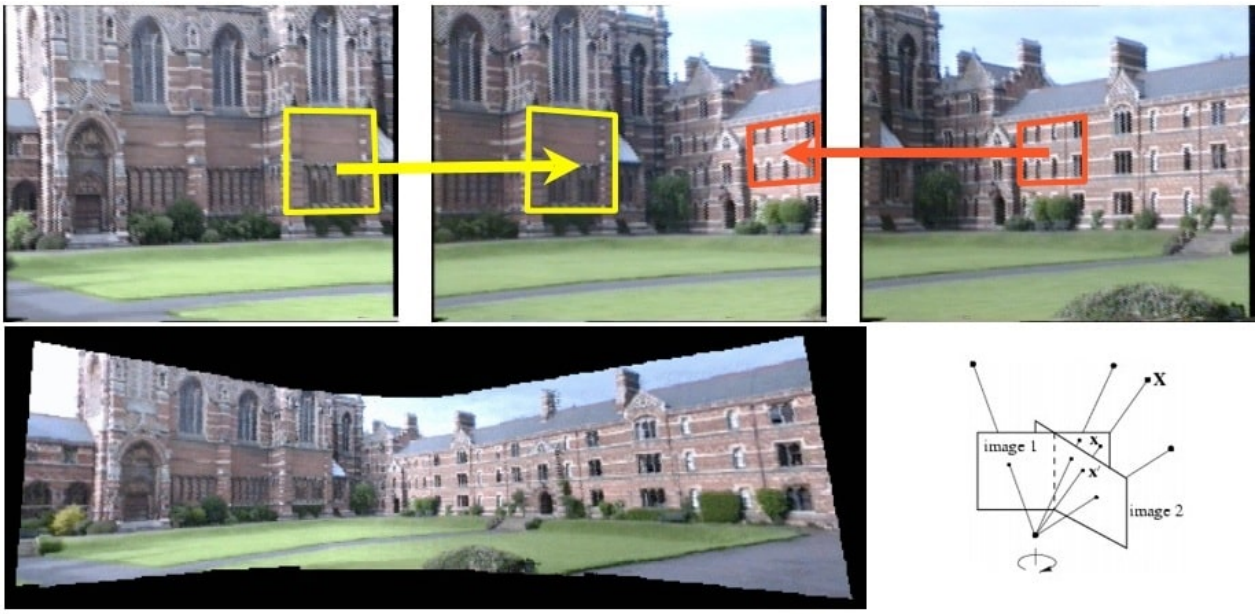
(b) algoritmus *FLANN*

Obrázek 4.6: Grafické zobrazení podobností po filtraci shod

Z vyhodnocených filtrovaných shod lze získat homografii mezi dvěma snímky. Nalezením shod se předpokládá nalezení společné oblasti. Výstupem výpočtu homografie mezi dvěma snímky je homografická matice popisující souřadnice společné oblasti ve vstupních snímcích, tedy ve snímku prvním i druhém. Velikost matice je 3×3. Jednotlivé body této matice se nazývají jako planární body projekce. Na jejím základě lze například určit, jak upravit snímek první, abychom jej mohli vložit do snímku druhého, a tak vytvořili například panorama. Dalším praktickým příkladem je nalezení objektu v druhém snímku, kdy první snímek reprezentuje vstupní masku, tedy objekt libovolně natočený, a snímek druhý obsahuje objekt vyfocený v prostoru. Z výstupní homografické matice lze zjistit souřadnice objektu v druhém snímku, a tak provádět detekci objektů. V neposlední řadě lze získat perspektivní pohled a po aplikování matice tuto perspektivu eliminovat a získat tak rovný objekt. [63, 64]



(a) Projekce společné oblasti do vstupních snímků [65]



(b) Získání panoramatického snímku

Obrázek 4.7: Grafické znázornění získání homografie a její aplikace [66, 67]

Získání matice je provedeno pomocí metody *findHomography()*, která pro výpočet potřebuje deskriptory obou snímků. Je tedy zapotřebí zpětně získat z filtrovaných shod deskriptory. Dalším volitelným parametrem je takzvaný *ransacReprojThreshold*, který určuje, kolik testů bude provedeno na výstupní matici, nežli bude vyhodnocena jako správná. Podmínkou pro vyhodnocení matice je dodání aspoň 4 deskriptorů v obou snímcích. Nicméně tyto hodnoty nemusí správně určit společnou oblast a proto je vhodné dodávat více bodů. Pro případ, kdy nejsou dodány 4 body, metoda *findHomography()* selhává a způsobuje pád programu. Proto je nutné zajistit podmíněné zavolání. Ukázku popisovaných procesů lze vidět ve výpisu 4.3.

```
std::vector< Point2f > obj;
std::vector< Point2f > scene;
cv::Mat H;

for (int i = 0; i < good_matches.size(); i++) {
   obj.push_back(keypoints_object[good_matches[i].queryIdx].pt);
   scene.push_back(keypoints_scene[good_matches[i].trainIdx].pt);
}

if (obj.size() > 4) H = findHomography(obj, scene, RANSAC);
else cout << "Not enough points to calc H!!!";
```

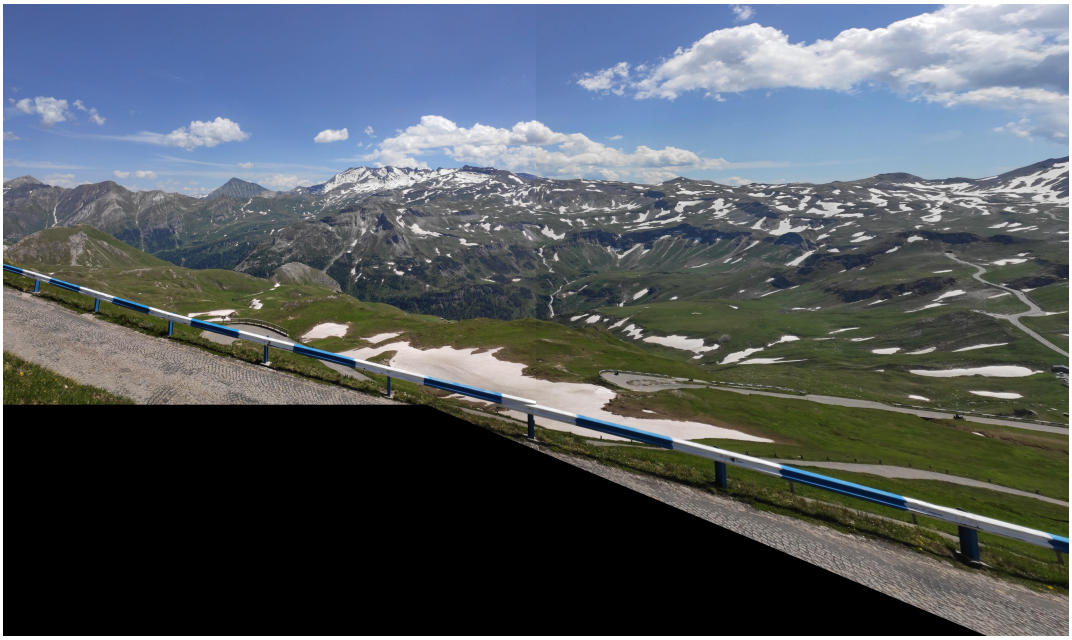
Výpis 4.3: Získání deskriptorů ze správných shod a vyhodnocení homografické matice

Aplikování perspektivy je provedeno metodou *warpPerspective()*, která jako vstupní parametry vyžaduje snímek, na který bude aplikovaná změna, reference na výstupní snímek, matice **H**, velikost výstupního snímku, a další konfigurační parametry pro výstupní snímek. Po aplikování metody na snímek získáme tento snímek posunutý v prostoru s aplikovanou perspektivou do výstupního snímku. Pro spojení dvou snímku je tedy zapotřebí pouze vložit snímek druhý do souřadnic 0, 0. Výsledkem je panoramatický obraz. Ukázku kódu je vidět ve výpisu 4.4. Na obrázku 4.8a je zobrazena aplikace perspektivy pomocí homografické matice metodou *warpPerspective()*. Výsledný obrázek je oříznutý z důsledku špatné úpravy snímku v dřívějších fázích.

```
Mat result;
warpPerspective(image1, result, H, cv::Size(image1.cols + image2.cols, rowsSize),
                                                    WARPindex, BORDER_TRANSPARENT);
Mat shift(result, cv::Rect(0, 0, image2.cols, image2.rows));
image2.copyTo(shift);
```

Výpis 4.4: Ukázka kódu pro spojení dvou snímků a vytvoření panoramatu

(a) Aplikování metody *warpPerspective()*



(b) Výsledný panoramatický obraz

Obrázek 4.8: Ukázka vytvoření panoramatického snímku

Kapitola 5

Návrh předvývojové elektroniky kamerového subsystému pro chytrý přední nárazník

Navrhované řešení představuje jednu z částí předvývojového chytrého nárazníku s názvem *SmartFace* (obrázek 5.1) společnosti *Hella* vývojového centra v *Berlíně* za pomoci společnosti *Plastic Omnium*. Chytrý přední nárazník je sestaven jako jeden plastový odlitek obsahující elektroniku, světlomety a senzory. Připojitelnost elektronické části nárazníků je realizovaná skrze jeden 4-pinový konektor, skládající se z napájení a komunikace. Dnešní automobily jsou doslova propleteny kabeláže mi. Výměna nárazníku moderního automobilu vyžaduje mnohdy odbornou montáž, kdy do nárazníku vede až jedna desítka konektorů. Chytrý nárazník přináší výhodu pro automobilku, která kupuje hotové řešení a nabízí snadnou montáž. Výměna takového dílu je rovněž snadná. Zodpovědnost za vady na výrobku nese dodavatel, automobilka se tedy stává pouze distributorem. V následující kapitole budou popsány požadavky na kamerový subsystém, výběr výpočetní techniky, vybrané kamery a vytvořená konstrukční řešení.

5.1 Požadavky kamerového subsystému

Chytrý nárazník obsahuje tři kamery nacházející se na stranách nárazníku vedle světlometů a uprostřed nárazníku nad logem. Z hlediska demonstračního účelu pro předvývojové účely byly definovány dodatečné požadavky. Nejdůležitějším požadavkem je přenositelnost kamerového subsystému, tedy poskytnout variabilitu záměny senzorické části a výpočetní techniky. Pro splnění tohoto požadavku je nutné použít kamery využívající ethernetové připojení nebo USB sběrnici. Neboť se prvky nachází v blízkém dosahu pár metrů od sebe, byly zvoleny kamery s USB rozhraním. Pro vývoj aplikace z počátku byla zvažována plně vestavěná aplikace. Podpora zpracování obrazu však na vestavěných platformách až pro tři kamery najednou nenalézá velkou podporu jak u programovacích kniho-

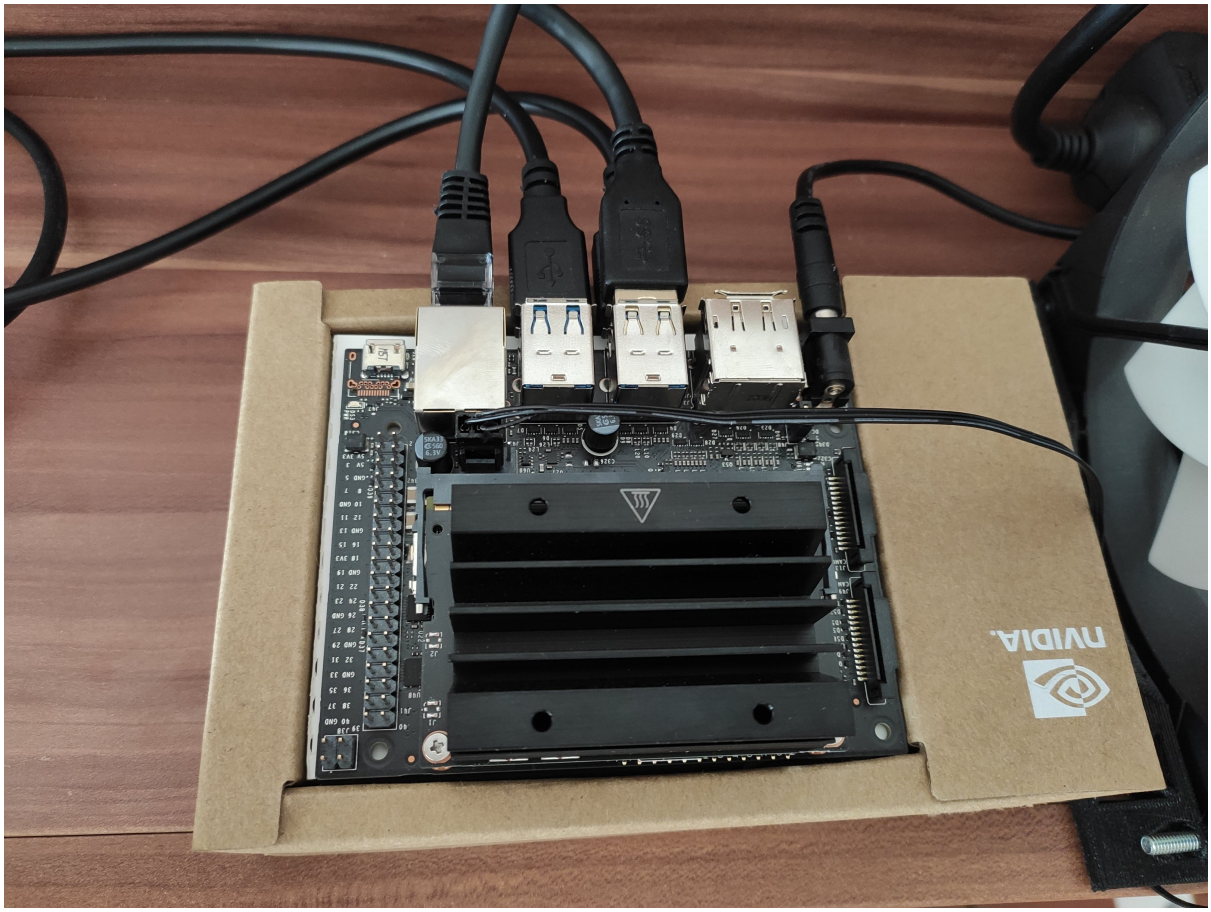
Obrázek 5.1: Chytrý přední nárazník *SmartFace*

ven, tak hotových vývojových desek. Byl tedy zvolen výběr mikrokontroléru s operačním systémem a napsání aplikace pro daný operační systém. Díky přítomnosti operačního systému a instalovaných knihoven je zajištěna přenositelnost programované aplikace. Druhým požadavkem je poskytnout živý přenos snímků z kamer při dostačující pořizovací kvalitě. Zvolené kritérium bylo rozlišení 1024×768 pixelů při 30ti snímcích za vteřinu, dosaženo však bylo mnohem víc, a to 1920×1080 pixelů při 30 snímcích za vteřinu. Třetím a posledním požadavkem je poskytovat panoramatický snímek složený ze tří snímků z kamer. Vyhodnocování panoramatu je zapotřebí zajistit z aktuálních snímků. Finální rozlišení ani počet snímků za vteřinu není specifikováno. Při pořizování komponent byl brán zřetel na co nejnižší pořizovací cenu a současně poskytnutí co největšího výkonu.

5.2 Výpočetní technika

Pří výběru výpočetní techniky je zapotřebí stanovit očekávání tohoto hardwaru. Pro řešení kamerového subsystému je zapotřebí dostačující výkon, zejména velikost paměti RAM. Cenově dostupnou variantou jsou často mikrokontroléry disponující pamětí RAM o 2 Gb, tato hodnota by však nemusela pro budoucí použití stačit. Proto byl zvolen požadavek velikosti paměti aspoň 4 Gb. Pro práci s obrazem je možné využít grafické akcelerace, tedy přítomnost takovéto komponenty je rovněž braná v potaz. Pro nutnost vyžívat kamery komunikující pomocí sběrnice USB 3.0 je nutné, aby mikrokontrolér měl takovýto řadič. Pro plynulý chod několika úkolů běžících paralelně v systému je vhodné vybrat vícejádrové CPU. Pro mikrokontroléry lze nalézt maximální počet jader 2 až 4. Vyšší počet jader je již málo častý a velmi finančně nákladný, pro využití vícero jader je už vhodnější pořídit dva mikrokontroléry a rozdělit operace na dvě části.

Vybraným mikrokontrolérem splňujícím stanovené požadavky je *Jetson Nano* od společnosti *NVIDIA*. MCU disponuje navíc zaměřením se na zpracování umělou inteligencí pracující s obrazem a zvukem. Vývojová deska disponuje 4-jádrovým procesorem ARM A57 o taktu 1,43 GHz. Dále grafickou akcelerací o 128 jádrech architektury Maxwell, velikosti paměti RAM 4 Gb s rychlostí až 25,6 Gb/s použité technologie LPDDR4. Deska nabízí připojení až 4 zařízení USB skrze jeden USB 3.1 řadič. Napájení lze realizovat buďto z USB vstupu, nebo z napájecího konektoru. Pokud deska vykazuje potřebu velkého množství energie (nad 1 A), je doporučeno použít napájecí konektor. K napájení je využit síťový adaptér s napětím 5 V poskytující výkon až 15 W. Navíc oproti požadavkům deska nabízí 40 výstupních GPIO pinů obsahujících komunikační sběrnice I^2C, I^2S, SPI a UART. Deska neobsahuje integrovaný bezdrátový Wi-Fi adaptér, je zde však možnost připojení externího skrze USB. *Jetson Nano* představuje platformu ARM64 s operačním systémem Linuxu Ubuntu, rozšířenou o API knihovny a ovladače pro správu a práci s periferiemi desky. Softwarová podpora pro vývoj umělé inteligence nachází velkou podporu ze strany společnosti *NVIDIA*. K dispozici jsou rovněž různé knihovny zabývající se využitím umělé inteligence přímo pro vývojovou desku *Jetson*. Vývojová deska *Jetson Nano* byla vydaná v roce 2020 s cílem nabídnout cenově dostupnou vývojovou desku. Díky tomuto kroku vzrostla rapidně komunita vývojářů. [68]

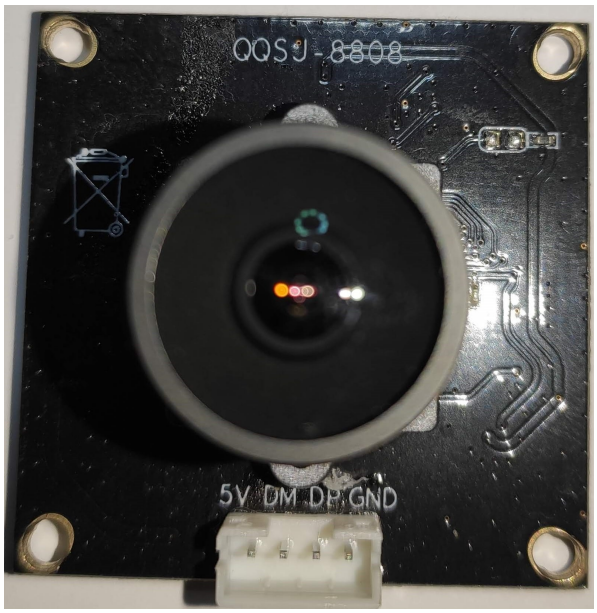


Obrázek 5.2: Vývojová deska *Jetson Nano*

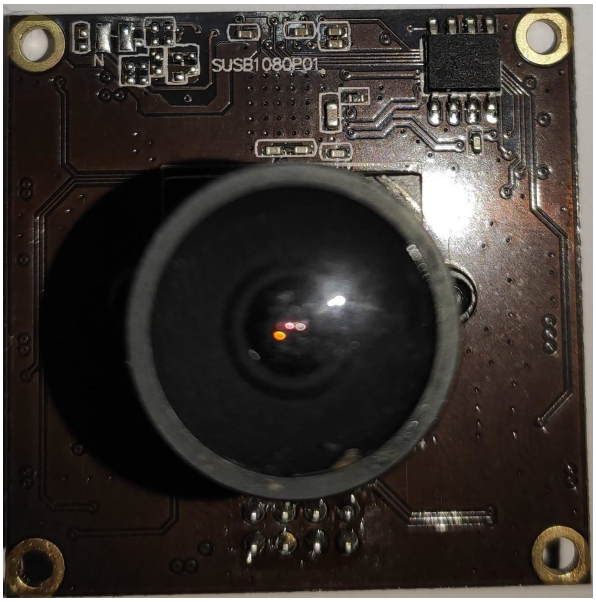
V rámci testování byl vyzkoušen mikrokontrolér *Raspberry Pi 4* verze 4 Gb paměti RAM, v této době novinkou na trhu z rodiny *Raspberry*. Pro testování výkonu zařízení byly spouštěny celkem 3 instance aplikace pro vysílání živého přenosu z USB zařízení do webového rozhraní. Po spuštění dvou instancí operační systém *Raspberian* zvládal přenášet data s občasným přerušením, které se projevovalo nedodáním některých snímků. Tato chyba se vyskytovala přibližně jednou za minutu. Po spuštění třetí instance byl již operační systém neschopen provozu a došlo k zamrznutí. Bylo tak způsobeno zahlcením USB sběrnice 2.0. Pro operační systém *Jetson Nano* byla tato chyba podchycena a došlo k zamezení přístupu třetí instance aplikací na sběrnici USB. Toto testování vedlo ke stanovení požadavků pro výběr mikrokontroléru.

5.3 Senzorická část – kamery

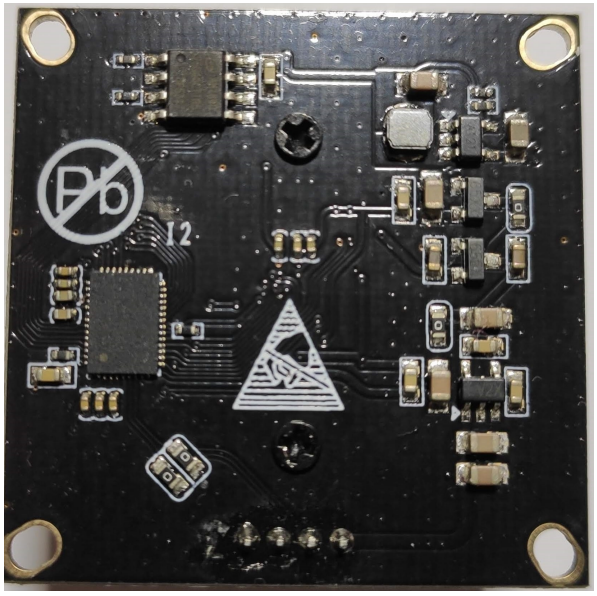
Kamer s podporou komunikace skrze USB je na trhu celá řada. Lze zakoupit hotové výrobky dostupné pro stolní počítače, známé jako web-kamery, nebo odhalené desky s elektronickými součástkami, snímačem a čočkou, známé jako kamerové moduly. Pro účely testování je zapotřebí, aby bylo možné kameru uchycovat pomocí šroubků. Záměna čočky je u kamerových modulů rovněž možná. Výběr je tedy směřován spíše k modulům. Pro finální řešení jsou zapotřebí dvě kamery využívající sběrnici USB 2.0 a jednu kameru komunikující po sběrnici USB 3.0. Případně lze kombinovat i tři USB 3.0 kamery. Zvoleným CMOS snímačem pro USB 2.0 je kamerový modul je *IMX179* od společnosti *Sony* schopen poskytnout 8mi megapixelové snímky o velikosti 3288×2512 pixelů. Zorný úhel kamerového modulu je 145°. Formáty předávání snímku skrze sběrnici USB jsou *MJPEG* a *YUY2*. V případě pořizování videa je modul schopen při Full HD rozlišení poskytnout až 30 FPS pro komprimovaný formát *MJPEG*, pro formát *YUY2*, který reprezentuje surová nekomprimovaná data jsou snímky za vteřinu již jen 3. Pořizovací cena je přibližně 1 100 Kč včetně DPH. [69] Výběr kamerového modulu s podporou USB 3.0 byl již o něco komplikovanější. Po takovýchto modulech nebyla doposud velká poptávka. Typickou kategorií tohoto kritéria jsou průmyslové kamery, které jsou vyráběny na zakázku. Pořizovací ceny jsou však vysoké a začínají na 7 000 Kč za kus. Na čínském trhu lze nalézt jediný kamerový modul se CMOS snímačem *IMX291* rovněž od společnosti *Sony*. Nabízené kamerové moduly jsou dostupné s čočkami s pozorovacím úhlem okolo hodnot 100°. Čočka byla dodatečně pořízena a vyměněna za shodnou s ostatními kamerovými moduly pro pokrytí pozorovacího úhlu 145°. Kamera se snímačem *IMX291* je schopna poskytnout dvoumegapixelové snímky o maximální velikosti 1945×1109 pixelů. Podporované komunikační formáty jsou *MJPEG* a *YUY2*. V případě videa je modul schopen poskytnout při rozlišení Full HD až 50 FPS při použití komprimace *MJPEG*. Pro surový formát *YUY2* pak 30 FPS. Pořizovací cena je přibližně 1 500 Kč včetně DPH.[70]



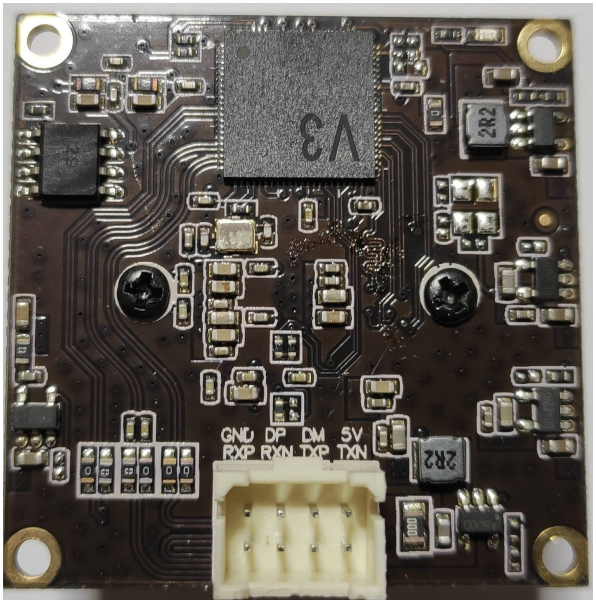
(a) (IMX179) Kamerový modul – přední strana



(b) (IMX291) Kamerový modul – přední strana



(c) (IMX179) Kamerový modul – zadní strana

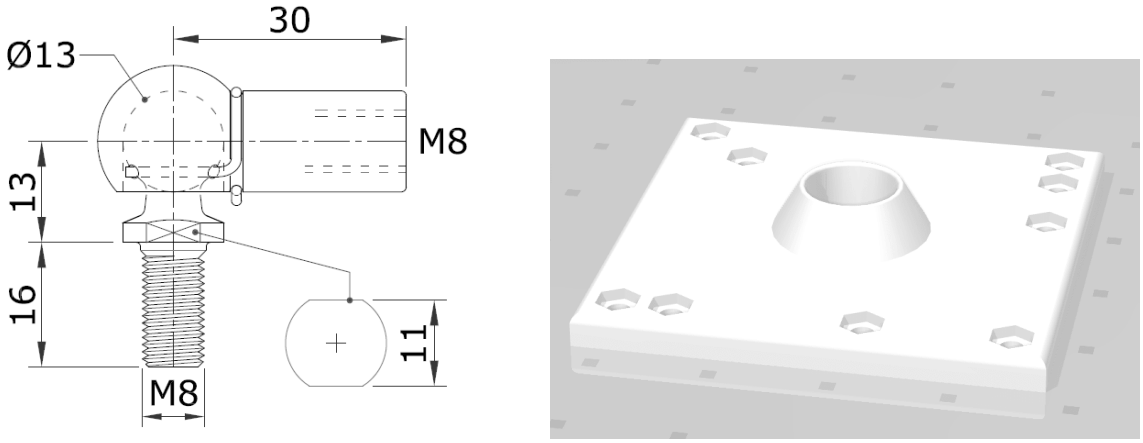


(d) (IMX291) Kamerový modul – zadní strana

Obrázek 5.3: Fotografie kamerových modulů z přední a zadní strany

5.4 Konstrukční části a výrobky

Pro uchycení kamer jsou vytvořeny podstavy a stojan, přičemž podstavy slouží pro vývoj obslužného softwaru v prostorách kanceláře, kde není možná přítomnost chytrého nárazníku, případně stojanu. Stojan je demonstrační náhražkou chytrého nárazníku, která umožňuje přichycení kamer a dalších senzorů. Ke stojanu je k dispozici box obsahující výpočetní techniku. Pro možnost libovolného natáčení kamer byl vybrán kulový kloub (Obrázek 5.4a) se závitem M8, kameru je tak možné rotovat v úhlu 360° pro horizontální pohled a naklánět a vyklánět v úhlech $-30°$ až 30°. Pro uchycení kamerového modulu byl vytvořen plastový adaptér vytisknutý 3D tiskárnou (Obrázek 5.4b). K vytvoření 3D modelů a výkresů byl využit software *AutoCAD* verze 2021.



(a) Výkres kulového kloubu [71] (b) Plastový adaptér k uchycení kamerového modulu

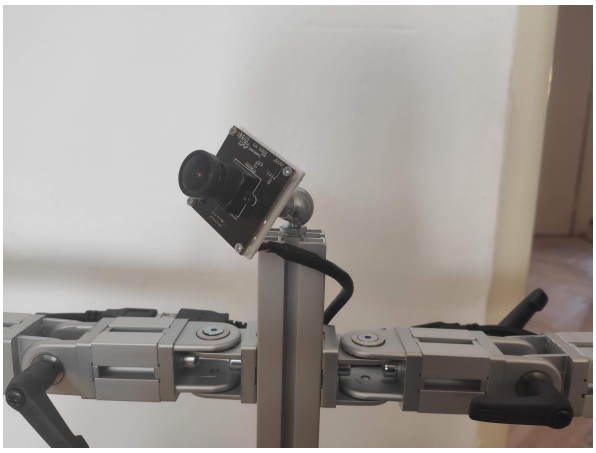
Obrázek 5.4: Kulový kloub a plastový adaptér

5.4.1 Stojan pro uchycení senzorů

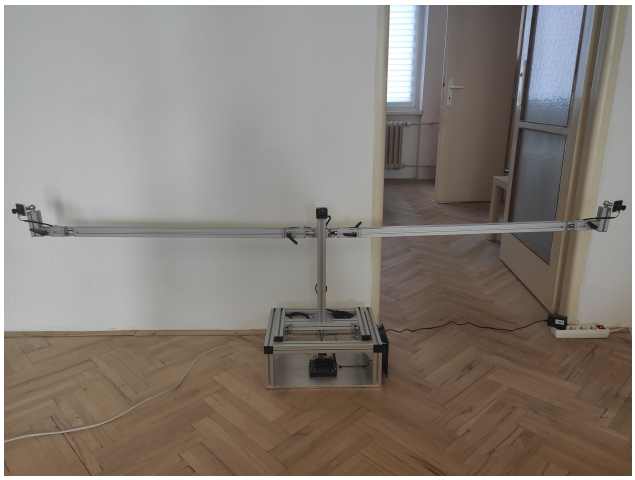
K simulaci poloh kamer byl vytvořen stojan sestavený ze dvou částí, který představuje šířku předního nárazníku vozidla. První část je tvořena stojanem. Dosahuje rozpětí až 1942 mm odpovídající šířce moderních limuzín. Nicméně umístění kamer v praxi je povětšinou v okolí světlometů nebo přímo uvnitř světlometů, bylo tedy počítáno s delším rozměrem. Hliníková konstrukce obsahuje čtyři klouby sloužící k polohování ramen ve směru vzhůru i do stran a k případnému zkracování celkové šíře. Ramena drží pomocí šroubů v hliníkových profilech. Kamery samotné jsou uchyceny v kulových kloubech. V kombinaci požadavku na zkrácení délky a zachování kamer v jedné rovině lze ramena uvolnit a uchytit na spodní straně svislého profilu. Tímto vznikne pomyslný trojúhelník a dojde ke zkrácení celkové šířky. Touto metodou lze získat minimální rozpětí 1535 mm. Druhou extrémní podmínkou je požadavek na plnou šíři ale s výškovým rozdílem kamer. K tomuto slouží hliníkové profily na konci ramen, které jsou rovněž uchyceny šrouby, a které lze uchytit v různých bodech profilu. Přípravek je modulární a připraven k budoucímu použití u vícero typů aplikací, jako

je například přidání jiných senzorických částí předního chytrého nárazníku. Noha stojanu je navržena tak, aby zapadla do střechy druhé části. Druhou část tvoří box vytvořený z hliníkových profilů, plexiskla a hliníkové desky. Slouží pro umístění výpočetní techniky a ostatních zařízení, jako je například ventilátor, kabeláž, napájecí adaptér a podobně. Základna, která jako jediná nebyla specifikovaná ve výkresu, je tvořena hliníkovou deskou o tloušťce 10 mm. Části jsou navrženy pro na sobě nezávislém použití. Výkresy lze nalézt v příloze A tohoto dokumentu.

(a) Prostor boxu



(b) Uchycení a rotace kamer



(c) Rozložený stojan

(d) Složený stojan

Obrázek 5.5: Fotografie stojanu

Konstrukce obsahuje i pár nedostatků. Zejména klouby držící konstrukci proti gravitaci vyžadují silné dotažení, aby nedocházelo k povolení a pádu ramen. Dále při rozhýbání stojanu nárazem dochází ke kmitání ramen v horizontální poloze. Stojan sám o sobě osciluje a k uklidnění dochází až po velmi dlouhé době. Je tedy zapotřebí asistence k urychlení ustálení. V neposlední řadě hliníková základna druhé části váží sama o sobě několik kilogramů a je z vodivého materiálu, což pro uchycení

odkryté výpočetní techniky není nejvhodnější základ, nicméně problém lze vyřešit aplikací izolační vrstvy.

5.4.2 Podstavy kamer pro vývoj

Pro vývoj softwaru a testování v malých prostorách je stojan nepraktický k použití. Z tohoto důvody byly navrženy podstavy pro jednotlivé kamery. Ty jsou tvořeny rovněž plastovým adaptérem, úhlovým kloubem, navíc se však skládají z hliníkové trubky a plastové podstavy. Na obrázku 5.6 lze vidět vytvořené podstavy pro jednotlivé kamery, rovněž na spodní části obrázku lze vidět kompletní rozpad dílů potřebných k sestavení.

Obrázek 5.6: Podstavy kamer pro vývoj

Kapitola 6

Návrh a realizace obslužného software kamerového subsystému předního nárazníku

Aplikace je tvořena čtyřmi procesy, tedy zpracováním snímků z kamer a jejich distribuce, získání panoramatického snímku, komunikace skrze websocket a webové rozhraní pro zobrazování výsledků. Díky požadavku pro demonstrační účely bylo možné využít vývojové desky s operačním systémem, jakými jsou například *Raspberian*, *Nvidia Ubuntu* a jiných na základě jádra *Unix/Linux*. Za přítomnosti operačního systému je možné využívat různé dostupné knihovny. Využitými externími knihovnami mimo operační systém jsou například *libjpeg-turbo* pro práci s daty z USB sběrnice, *Mongoose* a *POCO* pro správu webových serverů a *OpenCV* pro vyhodnocování panoramatického snímku. Vstupními požadavky je vyhodnocovat panoramatický snímek ze tří kamer, které musejí využívat rozhraní USB. Dále poskytovat dostatečné rozlišení a současně zajistit plynulost. Výstupní panoramatický snímek již plynulý chod poskytovat nemusí, nicméně musí být dodrženo zpracování v reálné čase, tedy nemělo by docházet ke zpracování již starých snímků a vždy zpracovávat ty nejaktuálnější. Na obrázku 6.1 lze vidět ukázku panoramatu vytvořeným algoritmem.

(a) Levý snímek

(b) Prostřední snímek

(c) Pravý snímek

(d) Vytvořené panorama z levého, prostředního a pravého snímku

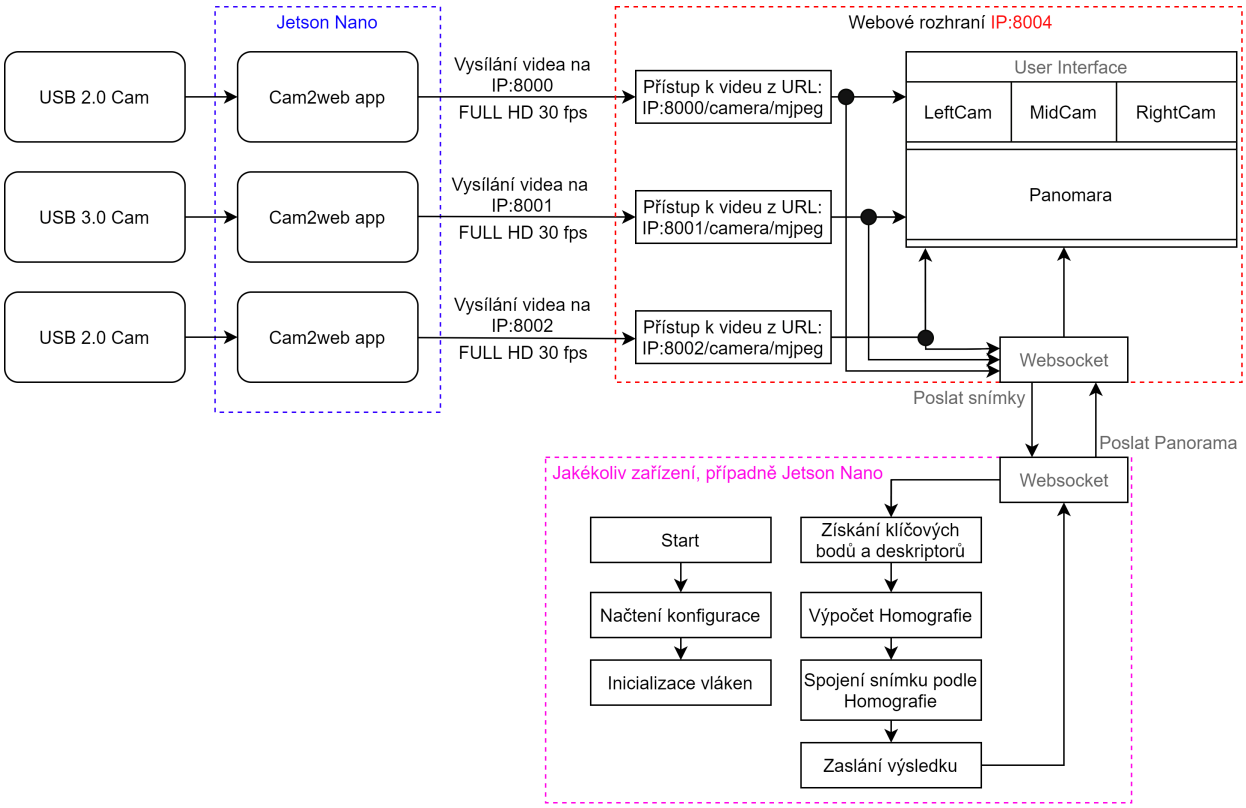
Obrázek 6.1: Ukázka vytvořeného panoramatického snímku aplikací pro vytváření panoramatu

Obslužný software je sestaven z několika programů nezávisle na sobě běžících. Závislost je tvořena pouze podmíněným předáváním dat. Členění aplikací je realizováno následovně:

- Aplikace pro zpracování snímků z kamer
- Aplikace pro vytváření panoramatických snímků
- Webové rozhraní

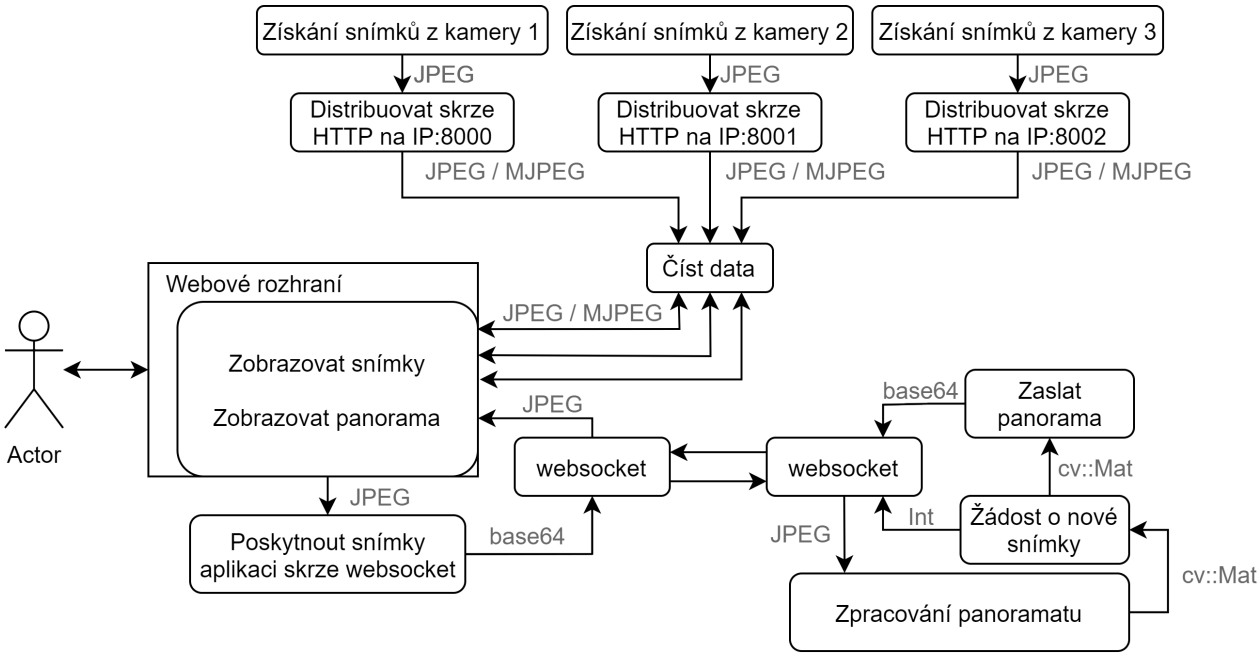
Zvoleným programovacím jazykem je zejména jazyk *c++* a webové jazyky jako *HTML*, *CSS* a *Javascript*. Aplikace pro zpracování snímků z kamer je převzatou open-source aplikací s názvem *Cam2Web*, která umožňuje vysílat přenos snímku z jednoho USB zařízení. Pro zajištění přenosu ze tří kamer jsou spuštěny tři instance této aplikace s různými vstupními parametry lišící se v definovaném portu a použitém zařízení. Aplikace vysílá pomocí HTTP protokolu a odesílané snímky je možné získat na adrese *IP:PORT/camera/mjpeg* nebo *jpeg*. Webové rozhraní za pomoci *Javascriptu* rozšířeného o knihovnu *Jquery* zobrazuje cyklicky snímky z definovaných adres do HTML prvků nacházející se na hlavní stránce. Pro získání panoramatického snímku je využit *websocket* podporován moderními prohlížeči. Snímek je přijat, následně zpracován a zobrazen do HTML prvku.

K získání vstupních snímků je rovněž využito navázaného oboustranného spojení skrze *websocket* mezi webovým rozhraním a aplikací pro zpracování panoramatu. Poté, co je panoramatický snímek vyhodnocen, je nejprve zažádáno o nové snímky a poté zaslán panoramatický snímek. Je tak dáno v důsledku vícevláknové komunikace. Popisované předávání informací a struktura aplikace je graficky znázorněna na obrázku 6.2. Aplikace lze přenášet na různá zařízení, podmínkou je společné síťové připojení.



Obrázek 6.2: Zjednodušený diagram funkcí a předávání dat mezi aplikacemi

Formát dat je v celém obslužném softwaru převážně ve formátu *JPEG*. V tomto formátu jsou již zpracovávány snímky z kamer. Podpora zobrazení *JPEG* je uplatněna i většinou prohlížečů. Formát využit pro přenos skrze *websocket* mezi webovým rozhraním a aplikací je *base64*. Tento formát nabízí kompaktní vyjádření dat a jeho celková velikost je výrazně menší. Je však zapotřebí provádět kódování a dekódování tohoto formátu. Pro aplikaci zpracovávající panorama je zapotřebí vstupní formát *JPEG* převést na formát *Mat*, který je součástí knihovny *OpenCV*. Formát *Mat* je maticový popis dat. Po získání výstupního panoramatického snímku je zapotřebí opět převod z formátu *Mat* na formát *JPEG*. Distribuce dat je popsaná na obrázku 6.3.



Obrázek 6.3: Diagram toků dat mezi aplikacemi

6.1 Aplikace pro zpracování snímků z kamer

Aplikace pro zpracování snímků z kamer je tvořena sběrem dat z periferie USB, základními filtracemi snímků a poskytováním živého přenosu na webové rozhraní skrze definovaný port. Aplikace nabízí konfiguraci pomocí parametrů při spouštění programu. Lze volit zařízení, velikost snímků, obnovovací frekvenci, vysílací port, specifikovat konfigurační soubor pro autentifikaci uživatelů, zvolit kořenový adresář k souborům webu, zvolit jméno aplikace a další. Zvolená rozlišení jsou vybrána za pomocí indexu, který je použit pro vybrání pozice v poli velikostí ve zdrojovém kódu. Jelikož je aplikace open-source, lze si například tyto hodnoty upravit a přizpůsobit aplikaci na míru. Aplikace je dostupná pro operační systém *Linux*, *Windows* a *Raspberian*, kde je využito knihovny pro sběr dat z CSI sběrnice. Pro operační systém *Linux* je zapotřebí instalovat dodatečné knihovny s názvem *libjpeg-dev*. Poté je možné aplikaci sestavit ze souborů *makefile* příkazem *make*. Webové rozhraní aplikace je dostupné na adrese ve formátu *IP:PORT*, kdy je javascriptem zobrazováno vysílání z adresy *IP:PORT/camera/mjpeg*. Některé aplikace formát *MJPEG* nepodporují, a tak je možno využít i vysílání jednotlivých snímků na odkazu *IP:PORT/camera/jpeg*, kdy je načten vždy nejaktuálnější snímek. [72]

```
./cam2web -help
./cam2web -size:7 -port:8000 -dev:0
./cam2web -size:7 -port:8001 -dev:1 -fps:20 -web:../mujweb - title:mujweb
```

Výpis 6.1: Příklady spouštěcích příkazů aplikace *Cam2Web*

Pro vytvoření aplikace pro zpracování snímků z kamer byly provedeny zkušební testy, které se zabývaly použitím různých knihoven na platformě *ARM*. Bylo zjištěno, že tyto knihovny mnohdy nepodporují komprimaci *JPEG* pro přenos dat po sběrnici USB a jsou schopny distribuovat pouze surová data ve formátu *YUY2*. To vedlo k razantnímu poklesu obnovovací frekvence zapříčiněného přenosem velkého množství dat. Problém se týkal zejména aplikace na platformě *ARM*. Na desktopových verzích se tento problém se stejnými knihovnami nevyskytoval. Testy byly prováděny na platformě *Raspberry Pi 4*. Výsledky pro rozlišení 1920×1080 byly 3 až 5 FPS. Pro menší rozlišení postupně počet snímků za sekundu rostl. Výsledky jsou závislé na použitém USB zařízení a na typu USB sběrnice. Řadič pro kamerové moduly s čipem *IMX179* využívají USB sběrnice 2.0 dosahující rychlosti až 60 MB/s, nicméně tato kapacita je pro formát *YUY2* dosažena již při rozlišení 800×600 pixelů a 30 FPS a dvou zařízeních. Samotný řadič je schopen dodat jen poloviční výkon a to 15 FPS při tomto rozlišení. Testované knihovny a jejich použití na platformě *ARM* s podporou komprimace *MJPEG* byla neúspěšná. Možným řešením je úprava dostupných knihoven pro platformu *ARM*. Takový zásah je však velmi rozsáhlý, přednost byla dána hotovým řešením, jakým je například aplikace *Cam2Web*.

Nalezená aplikace *Cam2Web* obsahující knihovny *Video4Linux* dostupné společně s aplikací podporovala sběr dat z USB zařízení ve formátu *MJPEG* i na platformách *ARM* a byla schopna poskytnout rozlišení 1920×1080 pixelů při 30 FPS. Pro otestování, zda se aplikace hodí pro zpracování snímků ze tří kamer, byly spuštěny nejdříve dvě instance této aplikace s různými parametry, poté tři instance. Na platformě *Raspberry pi 4* bylo možné spustit pouze dvě instance, neboť po spuštění třetí instance došlo k překročení fyzických vlastností sběrnice USB 2.0 a k zamrznutí systému. Na platformě *Jetson Nano* je systémově zajištěna přístupnost k periferiím a zabránění případným komplikacím. Při požadavku spustit třetí instanci je systémem zabráněno k přístupu na sběrnici USB. Je tak vyhodnoceno na základě přiřazení datových toků aplikacím a na základě jejich charakteru. Pro platformu *Jetson Nano* je možné použít nejvýše dva kamerové moduly komunikující skrze sběrnici USB 2.0. Pro využití třetí kamery bylo zapotřebí využít USB sběrnici 3.0. Aplikace *Cam2Web* úspěšně absolvovala živý přenos pro tři její instance při rozlišení 1920×1080 pixelů a při 30ti snímků za vteřinu. Zatížení hardwarových komponent bylo přibližně 30 % jednoho jádra CPU a využití přibližně 200 MB paměti RAM, aplikace nevyužívají grafické akcelerace.

Následující ideou bylo sjednotit aplikace a získávat snímky ze tří kamer najednou. Ty distribuovat na společný port s různými cestami, pro příklad *IP:PORT/camera1/mjpeg*, */camera2/mjpeg* a */camera3/mjpeg*. Po neúspěšných pokusech nabourat datový tok snímků, kdy byl proveden pokus o přepis definovaných pixelů, byly zváženy programátorské schopnosti a následky neprofesionálního zásahu do profesionálního řešení. Od idey upravovat tuto aplikaci bylo odstoupeno, neboť vhodnějším řešením by bylo aplikaci napsat celou znovu.

6.2 Aplikace pro vytváření panoramatického snímku

Aplikace na základě vstupních parametrů a snímků vypočte klíčové body, deskriptory a vzájemné homografické matice mezi snímky. Výstupem je panoramatický snímek, který může být distribuován pomocí *websoketu* na definovaném portu, nebo ukládán jako soubor na lokálním úložišti. Vstupní parametry jsou nastavovány před startem aplikace v hlavičkovém souboru *config.h*. Při vývoji aplikace byly vytvořeny tři třídy. Třída *trStitcher* slouží pro výpočet panoramatického snímku a obsahuje veškeré procesy potřebné pro jeho získání. Třída *measureTime* slouží pro měření časů v milisekundách a přehlednému výpisu. Třída *WebSocketServer* slouží pro vytvoření a správu *websocketu*. Aplikace je schopná zpracovávat libovolný počet snímků, doporučený počet je závislý na počtu jader CPU, neboť pro každý snímek je vytvořeno jedno vlákno. V aplikaci není optimalizován výpočet v závislosti na dostupných jádrech, je však definován maximální povolený počet vláken, a to na 8. Zvolené vývojové prostředí je *Visual Studio 2019* verze 142 využívající *Windows SDK* verze 10.0, aplikace je psaná v programovacím jazyku *c++* pro verzi systému x64 a operační systém *Windows 10*. Použitými externími knihovnami jsou *OpenCV* verze 4.5.1 a *POCO* verze 1.10.1. Aplikace využívá minimum systémových funkcí *Windows*, lze jednoduše upravit a přizpůsobit i pro operační systém *Linux*. Ke zdrojovým kódům aplikace jsou k dispozici UML diagramy a dokumentace.

6.2.1 Příprava knihoven a konfigurace projektu

Knihovny je možné použít na různých operačních systémech. Sestavení knihoven se provádí za pomoci nástroje *cmake*. Pro systém *Windows* je zapotřebí tento nástroj dodatečně nainstalovat. Před instalací platí všeobecný předpoklad přítomnosti potřebných systémových knihoven dostupných v rámci aktualizací systému. Dalším potřebným nástrojem sestavení knihoven ze zdrojového kódu je generátor či kompilátor *c++*, pro který nástroj *cmake* připraví soubory. Obvykle voleným generátorem pro systém *Windows* může být *Visual studio generátor*. Pro systém *Linux* lze sestavit knihovnu pomocí generátoru *make* ze souborů *makefile*. Sestavení ze zdrojových kódů je prováděno zejména kvůli nenáročnosti na velikost při přenosu a připravení knihoven na míru pro operační systém, kdy ne v každém systému jsou stejné nástroje.

6.2.2 Příprava knihovny *OpenCV* a *POCO*

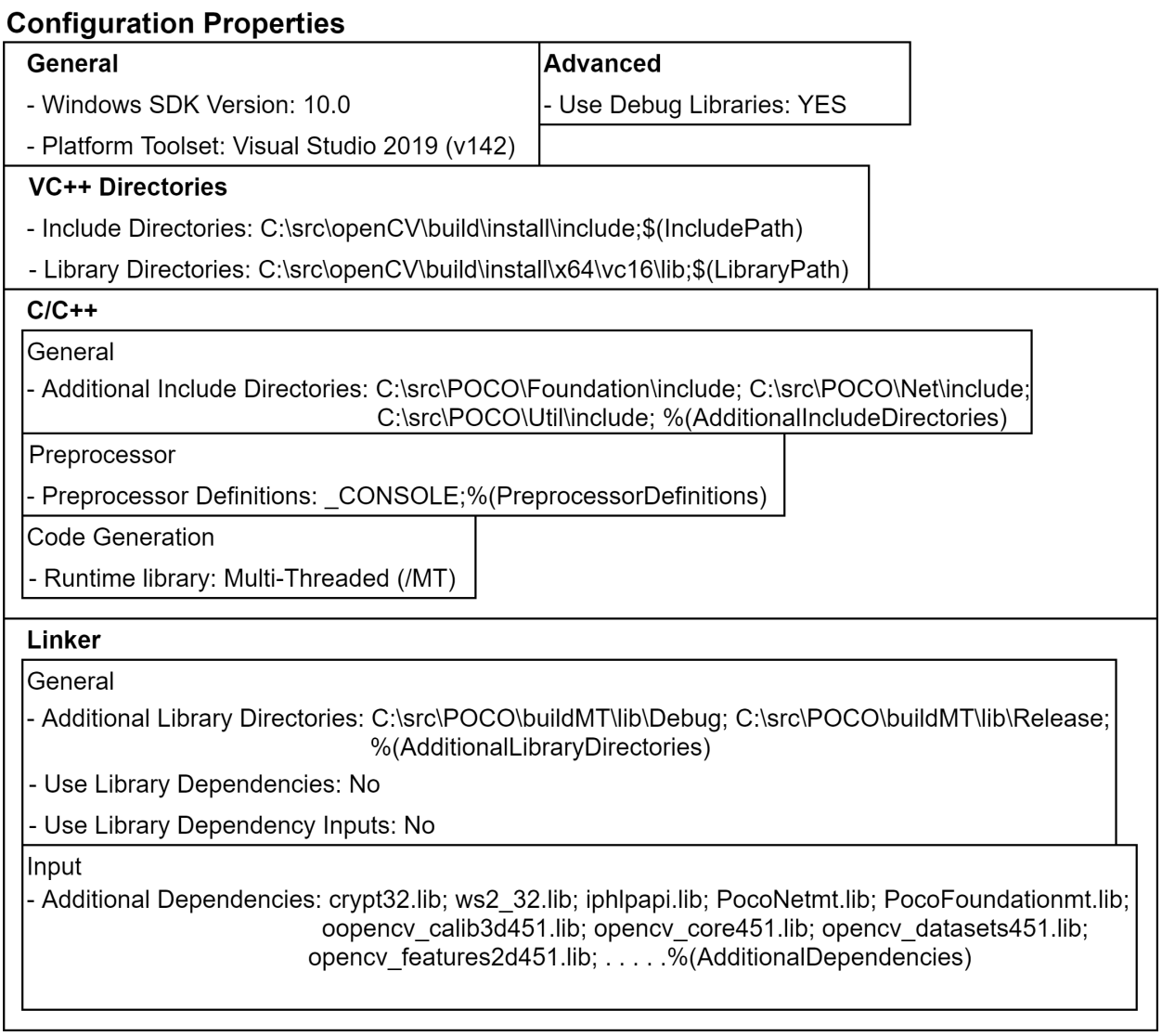
Pro využití dodatečných funkcí, jako je například získání klíčových bodů pomocí metody *SURF*, je zapotřebí dodatečné edice s názvem *OpenCV contrib*. Tato edice je podmíněna nekomerčním použitím, v opačném případě je zapotřebí zakoupit licenci. Zdrojové kódy knihovny *OpenCV* a edice *OpenCV contrib* lze získat ze serveru *Github*. Spuštění nástroje *cmake* lze provést dvěma způsoby, buďto jako zaslání příkazů skrze příkazový řádek k finální konfiguraci a následnému generování, anebo otevřením aplikace *cmake-gui*, což vede k otevření nástroje *cmake* s uživatelským rozhraním. V rozhraní lze vybrat vstupní parametry, a tak navolit správné příkazy pro *cmake*

generátor. Velikost zdrojových kódů knihovny *OpenCV* včetně příkladů je přibližně 325 MB. Velikost edice *OpenCV contrib* je přibližně 104 MB. Po sestavení knihoven pro typy ladění a uvolnění je velikost knihovny 28,5 GB. Čas potřebný pro sestavení knihovny je závislý na poskytnutém výkonu, pro procesor *Intel 6600K* s taktováním 4,4 Ghz a dostupných 16 GB paměti RAM trvá sestavení přibližně jednu hodinu pro oba typy knihoven. Na vývojové desce *Jetson Nano* trvá sestavení až 4 hodiny. Sestavení zabírá velké množství paměti RAM. Pro vývojovou desku *Jetson Nano* základní velikost paměti 4 GB nestačí. Před sestavením je zapotřebí nastavit virtuální paměť s pomocí systému *Linux* přibližně o 2 GB, tedy celkem na 6 GB. Virtuální paměť využívá místa úložiště, které je však schopno pracovat přibližně 250krát pomaleji než paměť RAM. Proto je čas tak rapidně prodloužen oproti desktopovému počítači. Nejdůležitějšími parametry, které je třeba definovat, jsou „OPENCV_EXTRA_MODULES_PATH" (slouží k určení cesty *OpenCV contrib* edice) a „OPENCV_ENABLE_NONFREE", který povolí využití licencovaných knihoven. Ve finální aplikaci je však využito metody *SIFT* a tento parametr není zapotřebí definovat. Zvolen byl z důvodů testování nejvhodnější metodou pro zpracování obrazu. Všechny ostatní parametry jsou ponechány základní hodnotě. Po sestavení knihoven je zapotřebí definovat systému cestu ke knihovnám, které bude následná sestavená aplikace využívat. K možnému programování ve *Visual Studiu* je zapotřebí sestavené knihovny uvést v konfiguraci projektu, kde je zejména důležité uvést cestu k hlavičkovým souborům a cestu ke souborům statických knihoven. V neposlední řadě je zapotřebí definovat *linkeru*, které konkrétní knihovny budou v projektu využity. Provádí se vyplněním seznamu s názvy knihoven.

Příprava knihovny *POCO* je obdobně realizovaná jako pro knihovnu *OpenCV*. Při konfiguraci *cmake* je však velmi důležité cílit sestavení pro budoucí aplikaci. Pokud aplikace bude využívat vícevláknové statické knihovny, je zapotřebí zvolit parametr „POCO_MT" a současně zrušit parametr „BUILD_SHARED_LIBS". Pokud aplikace bude využívat dynamické vícevláknové knihovny, je zapotřebí definovat pouze parametr „BUILD_SHARED_LIBS". Všechny ostatní parametry jsou ponechány základní hodnotě. Velikost sestavené knihovny je přibližně 980 MB a trvá několik minut, velikost zdrojových kódu knihovny i s ukázkami kódů je přibližně 930 MB. Problematičtější je implementace knihoven do prostředí *Visual studia 2019*. Obdobně jako u knihovny *OpenCV* je zapotřebí uvést cestu k hlavičkovým souborům a souborům knihoven. Poté definovat názvy knihoven, které budou v projektu použity, navíc je však nutné uvést i systémové knihovny, které knihovna *POCO* využívá. Potřebné systémové knihovny se liší podle využitých pod-knihoven. Pro vyvíjenou aplikaci byla využita třída *HTTPserver* a *ServerApplication*, které jsou součástí knihovny *POCO*. K úspěšnému sestavení vývojové aplikace a správné funkčnosti je zapotřebí, aby *linker* využil *POCO* knihovny *PocoNetmt.lib* a *PocoFoundationmt.lib*, navíc je třeba definovat systémové knihovny *crypt32.lib*, *ws2_32.lib* a *iphlpapi.lib*, bez kterých kompilátor nesestaví aplikaci. Tyto informace bohužel nejsou zdokumentovány a jsou často dostupné pouze z praxe ostatních vývojářů, kteří již obdobný problém řešili.

6.2.3 Příprava projektu v prostředí *Visual Studio 2019*

Ke správnému chodu aplikace a k jejímu sestavení bylo nezbytných několik změn v nastavení projektu. Protože každá knihovna vyžaduje jiné konfigurační hodnoty, bylo zapotřebí nalézt společnou konfiguraci. Správnou kombinaci nastavení popisuje obrázek 6.4, na kterém jsou uvedeny změny oproti základnímu nastavení při vytvoření projektu. Po definování hlavičkových souborů a cest k souborům knihoven je nejdůležitější úpravou změna hodnoty parametru „Runtime library" na „Multi-Threaded (/MT)", případně verze pro ladění, kdy se jedná o jedinou hodnotu, pro kterou fungují obě knihovny současně. Další důležitou úpravou pro knihovnu *POCO* je odstranění definice „_DEBUG" z parametru pro preprocesory definice v případě, kdy nebylo zvoleno využití sestavení pro ladění.



Obrázek 6.4: Konfigurační změny projektu ve *Visual Studio 2019*

6.2.4 Třída *MeasureTime*

Třída *MeasureTime* byla vytvořena pro měření uplynulého času dílčích částí programu a následnému přehlednému výpisu hodnot v milisekundách. Ve výpisu 6.2 lze vidět kompletní strukturu třídy. Pro její použití lze zvolit ze dvou konstruktorů, kde jediný rozdíl je v definování názvu instance. Pro započetí měření je nutné zavolat metodu *startMeasureTime()* s požadovaným určením názvu měření. Metoda přiřadí současný systémový čas do hodnoty *startTime* a přiřadí jméno měření. Pro ukončení měření je nutné zavolat metodu *stopMeasureTime()*. V této metodě je vypočten celkový čas a vytvořen záznam měření uložený do vektoru párů s názvem *Times*.

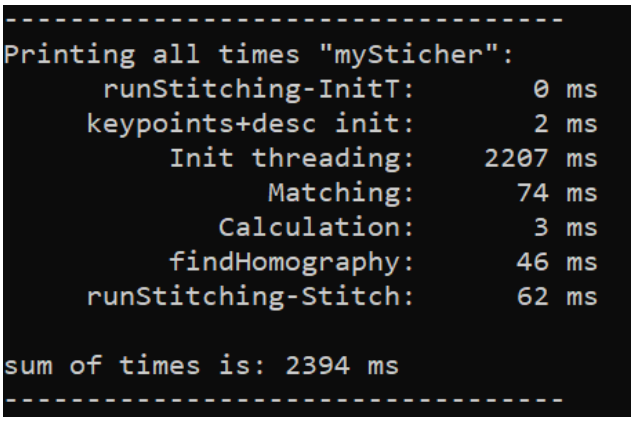
```
class measureTime {
    private:
        string measureTimeName;
        std::chrono::time_point<std::chrono::high_resolution_clock> startTime;
        std::chrono::time_point<std::chrono::high_resolution_clock> endTime;
        std::string nameOfTime;
        void setDefault();
    public:
        std::vector<std::pair<int, std::string> > Times;
        measureTime();
        measureTime(string name);
        void changeName(string name);
        void startMeasureTime(std::string name);
        void stopMeasureTime();
        void printAllTimes();
        void printNTime(int N);
        void printStrTime(std::string name);
        void printSum();
};

void measureTime::startMeasureTime(std::string name) {
    this->startTime = std::chrono::high_resolution_clock::now();
    this->nameOfTime = name;
}

void measureTime::stopMeasureTime() {
    this->endTime = std::chrono::high_resolution_clock::now();
    int result = std::chrono::duration_cast<std::chrono::milliseconds>(this->endTime - this->startTime).count
        ();
    this->Times.push_back(std::make_pair(result, this->nameOfTime));
    this->setDefault();
}
```

Výpis 6.2: Struktura třídy *MeasureTime*

K přehlednému výpisu je možné použít několik metod. Pro vypsání N-tého měření slouží metoda *printNTime()*. Pro vypsání měření s konkrétním názvem slouží metoda *printStrTime()*. Pro vypsání všech měření slouží metoda *printAllTimes()*. Pro přehledné vypsání všech měření s ohraničením a původním názvem instance slouží *printSum()*, jehož ukázku lze vidět na obrázku 6.5. Navíc je vypočtena a vypsána celková doba z dostupných měření.



Obrázek 6.5: Ukázka přehledného výpisu třídy *MeasureTime* metodou *printSum()*

Použití třídy je velice jednoduché (lze vidět ve výpisu 6.3). Je vytvořena třída *MeasureTime* s názvem instance „Toto je muj timer". Poté je zavolána metoda *startMeasureTime()* s názvem „JmenoMereni". Po vykonání algoritmu (úseku, který je měřen) je zavolána metoda *stopMeasureTime()*. Pro výpis tohoto měření je zvolena metoda *printSum()*.

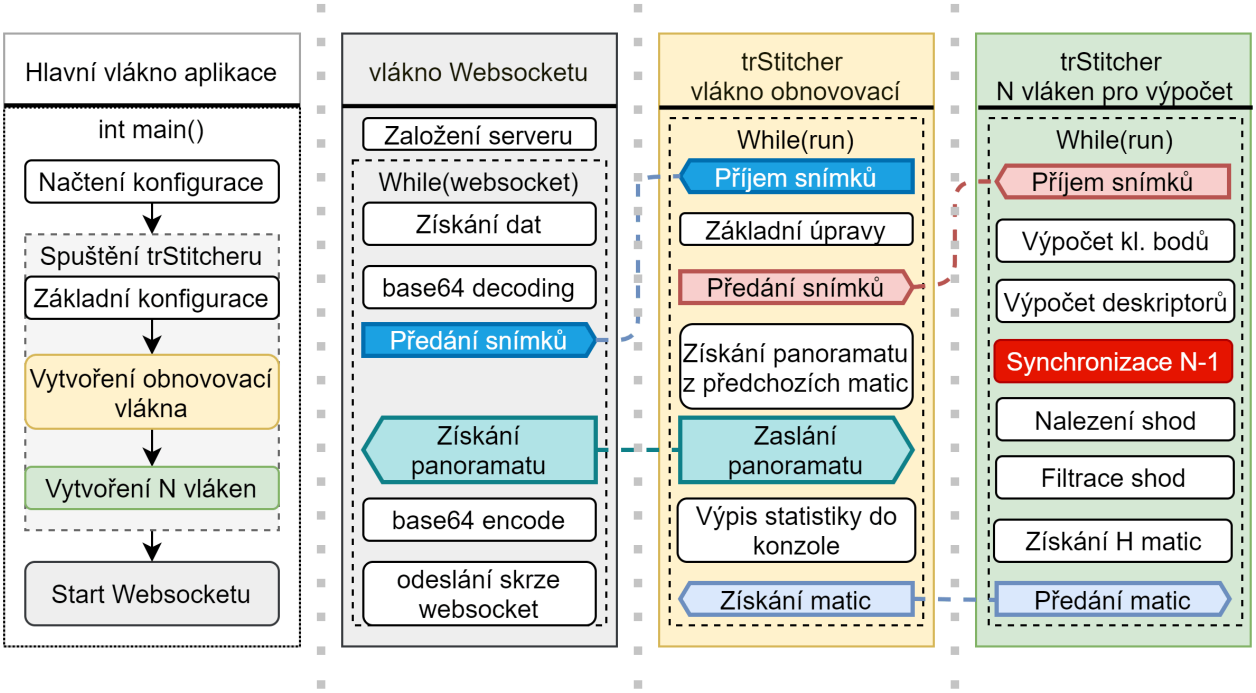
```
measureTime* timer = new MeasureTime("Toto je muj timer");
timer->startMeasureTime("JmenoMereni");
// . . . algoritmus . . . //
timer->stopMeasureTime();
timer->printSum();
```

Výpis 6.3: Použití třídy *MeasureTime*

6.2.5 Popis aplikace z hlediska vláken

Třída *trStitcher* na první pohled provádí jednoduchou operaci, tím je příjem snímků z webového rozhraní, jejich zpracování a vrácení panoramatického snímku zpět. K rychlému výpočtu je využito vícevláknového zpracování, a tak je tímto aplikace o dost rozsáhlejší a komplikovanější. Třída *trStitcher* disponuje volbou konfiguračních parametrů, jakými jsou například určení škálování snímků, určení škálování panoramatu, definování zpracovávání v černobílém režimu, definování poměrové hodnoty pro filtraci shod a další. Všechny konfigurační parametry mají vliv na finální výpočetní dobu. Součástí třídy jsou dva algoritmy pro vyhodnocení finálního panoramatu ve vícevláknovém provedení. V následujících odstavcích je vysvětleno vícevláknové řešení aplikace.

Na obrázku 6.6 lze vidět rozdělení aplikace na několik vláken. V hlavním vlákně je provedena základní rutina potřebná pro spuštění dalších vláken a k zahájení algoritmů tvořících aplikaci. Po načtení základní konfigurace je zavolána metoda instance třídy *trStitcher* pro započetí. V této metodě je spuštěno vlákno pro obnovování snímků a N vláken pro výpočty potřebné k získání panoramatu. Hodnota N je určena na základě počtu vstupních snímků. Tento počet musí být definován ve fázi konfigurace, tedy nelze jej měnit za chodu programu. Jako poslední krok hlavního vlákna je vytvoření HTTP serveru, který rovněž sleduje aktivity *websocketu*. Webový server běží ve vlastním vlákně. Třída pro HTTP server je naprogramovaná tak, aby blokovala hlavní vlákno a to se nikdy neukončilo. Pokud by takto nebylo učiněno, hlavní vlákno by skončilo a i všechna příbuzná vlákna programu.



Obrázek 6.6: Rozdělení aplikace na jednotlivá vlákna a jejich operace

Po příchodu požadavku skrze *websocket* je vytvořeno separované vlákno, které naváže komunikaci s klientem. V tomto vlákně je realizován přenos dat z webového rozhraní, tedy třech snímků. Po přijmutí všech paketů všech tří snímků a dekódování formátu *base64* na formát *JPEG* jsou snímky odeslány skrze společné globální proměnné vláknu pro obnovování. Vlákno dále čeká na příchod panoramatického snímku, který je po získání *enkódován* do formátu *base64* a odeslán skrze *weboscket* webovému rozhraní.

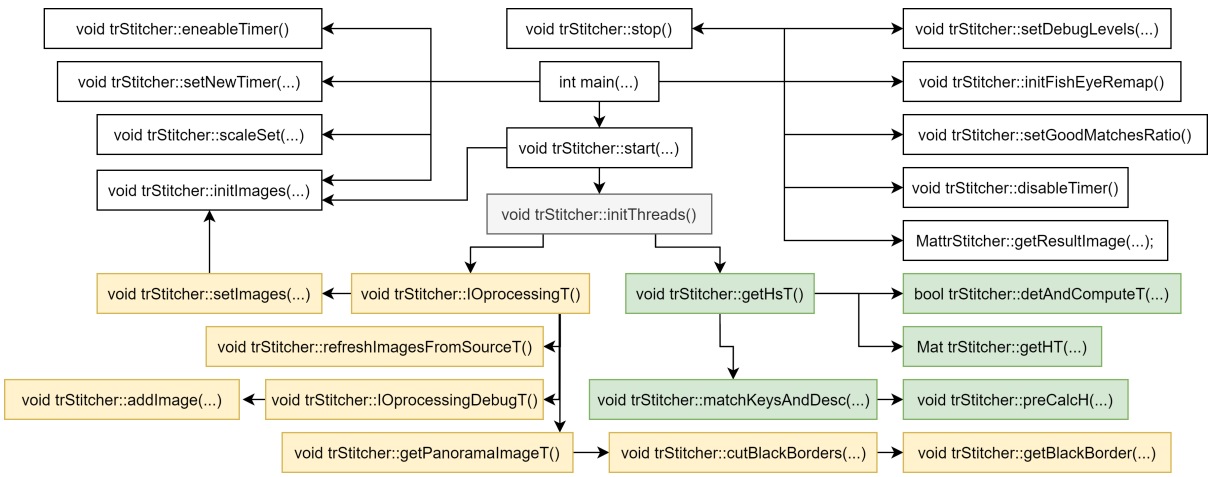
Vlákno pro obnovování je sestaveno ze smyčky *while*. Ve smyčce je nejdříve zjištěna dostupnost nových snímků. Pokud jsou nové snímky k dispozici, provedou se základní úpravy, jakými jsou výpočet černobílých snímků a kompenzace efektu rybího oka. Snímky jsou následně odeslány vláknům

pro výpočet matic skrze společné proměnné nacházející se uvnitř třídy. Aby vlákno pro obnovování nezůstávalo nečinné do doby, než budou vypočteny matice **H** všech snímků (což trvá několik desítek milisekund), je vyhodnocen panoramatický snímek z předchozích výsledků. Ten je následně zaslán vláknu s vytvořeným *websocketem* a navázaným spojením. V případě nastavení proměnných konfigurace pro ladění jsou do konzole tisknuty časy výpočtů a spojení snímků.

Vlákna pro výpočet získají ze vstupních snímků klíčové body a deskriptory. Každé vlákno pracuje jako „worker", tedy všechna vlákna jsou totožná a liší se pouze přiřazeným identifikačním číslem 0 až N, které je definováno algoritmem na začátku výpočtů. Na základě tohoto ID je převzat snímek a výpočty s ním spojené zařazované do polí či vektorů hodnot s tímto ID.. Po výpočtu deskriptorů je zapotřebí nalézt shody mezi dvěma snímky, provést filtraci shod a získat matici H. Je provedena synchronizace mezi těmito N vlákny, kdy každé vlákno převezme hodnoty vlákna následujícího, tedy $ID + 1$. V případě, že se jedná o vlákno poslední ($ID = N$), toto vlákno nemá další úkoly a vrací se zpět na začátek smyčky *while*, kde čeká na nový snímek. Probíhající výpočty jsou nyní vykonávány vlákny o počtu $N - 1$. Po vypočtení všech matic **H** jsou matice předány zpět vláknu obnovování.

6.2.6 Popis významných metod a proměnných třídy *trStitcher*

Pro přehlednost ve třídě *trStitcher* byla vytvořena mapa závislostí metod třídy (obrázek 6.7), která popisuje zdroje volání metod. Při vytváření metod byla brána v potaz co možná nejlepší přehlednost, proto každá metoda obsahuje pouze pár řádků kódu. Kód je tak čitelný a srozumitelný. Žlutou barvou jsou vyznačeny metody sloužící vláknu obnovování, zelenou barvou jsou značeny metody pro vlákna provádějící výpočet. Parametry metod nejsou uváděny v důsledku zachování přehlednosti mapy. Pokud metoda parametry obsahuje, jsou nahrazeny třemi tečkami v závorce metody. V následujících odstavcích budou popsány významné metody a proměnné.



Obrázek 6.7: Mapa závislostí metod třídy *trStitcher*

Třída *trStitcher* je tvořena jedním hlavičkovým souborem a čtyřmi zdrojovými soubory obsahujícími definice metod. Pro inicializaci základních hodnot je využíváno metody *setImages()* a *scaleSet()*. Metoda *scaleSet()* nastaví poměr škálování vstupních snímků a poměr škálování výstupního panoramatu. Pro započetí vláken je dostupná metoda *start()* s možností volby černobílého zpracování snímků. Metoda *stop()* slouží pro ukončení vláken a uvolnění paměti. Důležitou poměrovou hodnotou je rovněž proměnná *goodMatchRatio* definovaná metodou *setGoodMatchesRatio()*. K ukládání dat snímků jsou k dispozici proměnné *inputImages*, *images*, *imagesStepBack* a *outputImage*. Metoda reprezentující vlákno pro obnovování se nazývá *IOprocessingT()*, název metody reprezentující N vláken pro výpočty je *getHsT()*. Lze si všimnout použití písmena T na konci metod, které jsou vykonávány ve vláknech. Jmenované metody a proměnné jsou vidět ve výpisu 6.4.

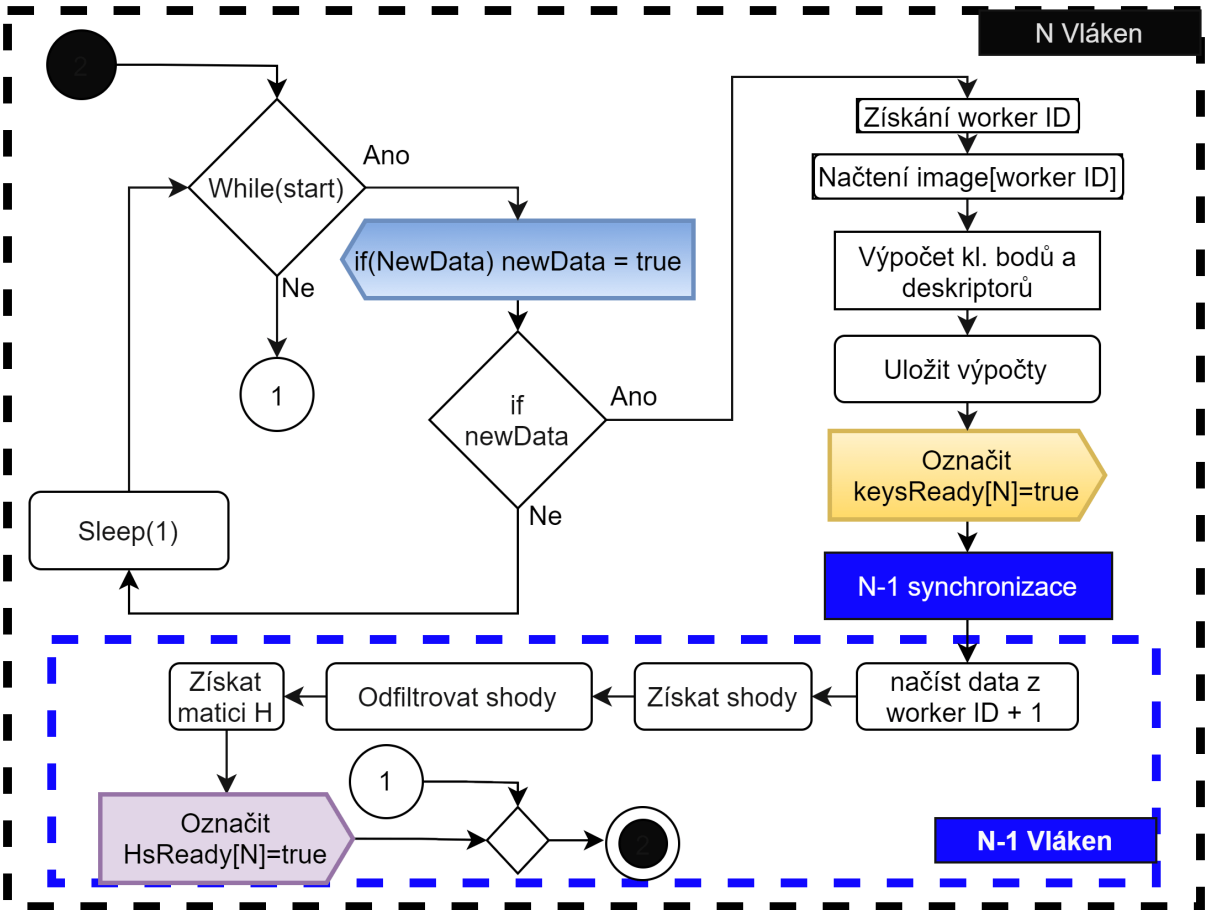
```
class trStitcher {
  public:
    void setImages(Mat images[], int numberOfElemets);
    void scaleSet(double processingScale, double outputScale);
    void setGoodMatchesRatio(double ratio);
    void initFishEyeRemap();
    void start(bool greyscaled);
    void stop();
    void initThreads();
    void IOprocessingT();
    void getHsT();
    . . .
  private:
    double inputResize;
    double outputResize;
    double goodMatchRatio;
    measureTime* timer;
    std::vector<Mat> inputImages;
    std::vector<Mat> images;
    std::vector<Mat> imagesStepBack;
    Mat outputImage;
    . . .
}
```

Výpis 6.4: Významné hodnoty a metody třídy *trStitcher*

6.2.7 Popis algoritmu pro výpočet matice H

Algoritmus je shodný pro všech N vláken, tudíž běží N-krát a je zapotřebí čelit kolizím a poskytnout synchronizaci mezi vlákny, které mají za úkol vypočítat homografickou matici **H** mezi dvěma snímky. Diagram aktivit je znázorněn na obrázku 6.8. V první fázi je zapotřebí detekovat, zda jsou dostupná nová data. Děje se tak skrze proměnou *NewData*. Jakmile je detekovaná hodnota *true*, pokračuje se k přiřazení ID vlákna. Následuje získání klíčových bodů a deskriptorů pro konkrétní snímek. Do

pole hodnot *keysReady* datového typu *bool* je přiřazena hodnota *true* s umístěním ID.. Následuje první synchronizace, která má za úkol převzít data z následujícího vlákna, tedy $ID+1$. Jakmile má algoritmus přístup ke klíčovým bodům a deskriptorům dvou snímků, lze přistoupit k získání shod, jejich filtraci a výpočtu matice **H**. Po jejím získání je hodnota příslušného pole s umístěním ID a s názvem *HsReady* nastavena na hodnotu *true*. V obrázku 6.8 je část vývojového diagramu společná pro všechny vlákna vyznačena v černém přerušovaném ohraničení. Část společná pouze pro $N-1$ vláken se nachází v modrém přerušovaném ohraničení.



Obrázek 6.8: Diagram aktivit pro získání matice **H**

Během výpočtů je využita struktura s názvem *detectorData* obsahující veškeré informace o vypočtených datech. Těmi jsou snímek, klíčové body, deskriptory a matice **H**. Tato struktura je však potřebná pouze pro přehledné předávání dat mezi vlákny. Pro následné spojení snímků postačí znát hodnoty matic **H** a původní snímky. Jak již bylo patrné z obrázku 6.7, použitými metodami pro získání matice **H** jsou *detAndComputeT()* a *getHT()*, která následně využívá další metody *matchKeysAndDesc()* a *preCalcH()*. Ve výpisu 6.5 lze vidět definici metody *detAndComputeT()* a struktury nutnou k předávání dat. Využitý je detektor *SIFT* z důvodu poskytnutí dostatečného

počtu klíčových bodů a současně korektních. V případě, kdy z jakéhokoliv důvodu selže detekce, je návratová hodnota funkce *false*.

```
struct detectorData {
        Mat image;
        std::vector< KeyPoint> keypoints;
        Mat descriptors;
        Mat resH;
};

bool trStitcher::detAndComputeT(trStitcher::detectorData* dataT) {
   try {
      Mat image;
      if (this->greyCalc) cv::cvtColor(dataT->image, image, cv::COLOR_BGR2GRAY);
      else image = dataT->image;
      // SIFT
      Ptr<SIFT> detector = cv::SIFT::create();
      detector->detectAndCompute(dataT->image, Mat(), dataT->keypoints, dataT->descriptors);
   } catch (cv::Exception& ex) {
      std::cout << ex.msg.c_str() << std::endl;
      return false;
   }
   return true;
}
```

Výpis 6.5: Proměnné pro synchronizaci mezi vlákny

Vyhodnocení matice **H** na základě dat ze dvou snímků se provádí metodou *getHT()* (k vidění ve výpisu 6.6). Vstupními parametry jsou dvě struktury *detectorData*. Zbývající proměnné potřebné pro výpočet jsou již lokální a dočasné. Metoda *matchKeysAndDesc()* využívá nalezení shod pomocí algoritmu *FLANN*, zvolen je zejména díky vysoké rychlosti vyhodnocení oproti algoritmu *BruteForce*. Vyhodnocené shody jsou vráceny pomocí parametru *matches*. Dalším krokem je filtrace nalezených shod prováděna metodou *preCalcH()*. Vstupními parametry jsou opět dvě struktury *detectorData*, dále vektor bodů s názvem *obj* a *scene*. Posledním krokem je získání matice **H** zavoláním metody *findHomography()*. Vstupními parametry jsou vektory bodů *obj* a *scene*, tvořeny filtrovanými shodami mezi snímky. Nachází se zde i podmíněné zavolání, neboť pro případ, kdy nejsou dodány aspoň 4 body, metoda *findHomography()* selže a program padá. Návratová hodnota je buďto nalezená matice **H**, nebo její nulová hodnota.

```
void trStitcher::matchKeysAndDesc(Mat& descriptorsObject, Mat& descriptorsScene,
                                  vector<std::vector<DMatch>>& matches) {
   Ptr<FlannBasedMatcher> matcher = cv::FlannBasedMatcher::create();
   matcher->knnMatch(descriptorsObject, descriptorsScene, matches, 2);
}
Mat trStitcher::getHT(detectorData& data1, detectorData& data2) {
   vector<std::vector<DMatch>> matches;
```

```
    this->matchKeysAndDesc(data1.descriptors, data2.descriptors, matches);
    std::vector< Point2f > obj;
    std::vector< Point2f > scene;
    this->preCalcH(data1, data2, matches, obj, scene);

    // Find the Homography Matrix for img 1 and img2
    cv::Mat H;
    if (obj.size() < 4) {
        if (this->debugLevel[2]) cout << endl << "!!!!!! not enough similarities (less than 4) !!!!!!" << endl;
    }
    else {
        H = findHomography(obj, scene, RANSAC);
        data1.resH = H;
    }
    matches.clear();
    obj.clear();
    scene.clear();
    return H;
}
```

Výpis 6.6: Metoda *getHT()* a *matchKeysAndDesc()*

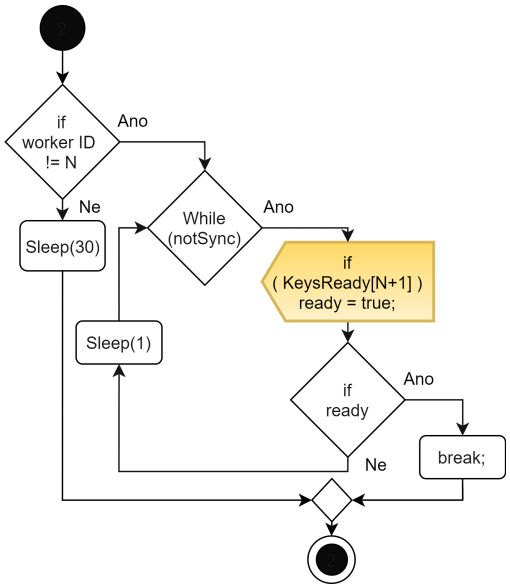
6.2.8 Popis algoritmu pro synchronizaci vláken

Důležitým faktorem pro realizaci synchronizace je definování ID jednotlivým vláknům při započetí požadavku na vyhodnocení matic **H**. Přiřazení ID je realizováno skrze společné počítadlo, kdy každé vlákno ve chvíli, kdy přistupuje ke snímkům, využije nástroj *mutex* a uzamkne počítadlo. Během tohoto uzamknutí přiřadí svému procesu ID na základě aktuální hodnoty počítadla. Následuje inkrementace samotného počítadla a jeho uvolnění. Algoritmus přiřazování ID vláknům je vidět ve výpisu 6.7.

```
if (newData) {
    MnewDataMutex.lock();
    workerID = MworkerCounter;
    MworkerCounter++;
    if (MworkerCounter > Hsize) {
        MworkerCounter = 0;
        MnewImagesAvaible = false; // RESET požadavku pro výpočet
    }
    MnewDataMutex.unlock();
    if (!MkeysReady[workerID]) CalcRequested = true;
    Mdata[workerID].image = this->images.at(workerID); // vyčtení snímku s workerID
}
if(CalcRequested) {. . .} // pokracovani algoritmu podmineno CalcRequested
```

Výpis 6.7: Přiřazení ID vláknům pomocí počítadla

K synchronizaci je využito nástroje *mutex*, kdy pro každé vlákno náleží právě jeden. Vhodnějším postupem je použití nástroje *barrier*, toto řešení však nebylo pro současnou aplikaci implementováno. Synchronizace s pomocí nástroje *mutex* může trvat až 5 ms navíc, je tak dáno smyčkou *while*, která obsahuje funkci *Sleep* o hodnotě 1 ms. Synchronizace vláken je prováděna dvakrát během vyhodnocování matice **H**. První synchronizaci je zapotřebí provést mezi sousedními vlákny, tedy vláknem ID a $ID + 1$. V případě, že se jedná o vlákno, kdy je N rovno ID, toto vlákno již synchronizováno není, jeho operace skončily a vrací se na začátek smyčky *while*. Diagram aktivit první synchronizace je znázorněn na obrázku 6.9.



Obrázek 6.9: Diagram aktivit pro synchronizaci dvou vláken

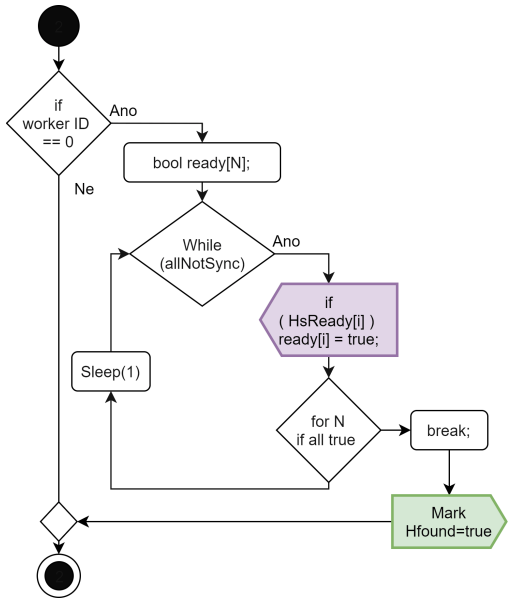
Zdrojový kód první synchronizace se nachází ve výpisu 6.8. Hodnota *Hsize* je rovna hodnotě $N - 1$ vláken. V prvním řádku je eliminováno poslední vlákno. Následuje smyčka *while* nahlížející k výsledku sousedního vlákna. Jakmile jsou data dostupná, smyčka *while* je ukončena a data jsou převzata. Na data sousedního vlákna již není zapotřebí nástroj *mutex*, neboť od této chvíle se do dat neprovádí zápis, pouze čtení.

```
if (workerID < Hsize) {
   int worker2 = workerID + 1;
   bool worker2Done = false;
   // Sync with ID+1 thread
   while (!worker2Done) {
      MgetKeysMutex[worker2].lock();
      worker2Done = MkeysReady[worker2];
      MgetKeysMutex[worker2].unlock();
      if (!worker2Done) Sleep(1);
   }
   data2 = Mdata[worker2];
```

. . .

Výpis 6.8: Zdrojový kód první synchronizace

Po vyhodnocení všech matic **H** všemi vlákny je zapotřebí matice **H** vhodně předat vláknu pro obnovování, kde je uskutečněno spojení snímků. Pro tento způsob je vytvořena druhá synchronizace, která po zjištění výpočtu všech matic označí příslušnou proměnnou *hFound* jako *true*. Tato hodnota je detekována ve vlákně pro obnovení. Na obrázku 6.10 lze vidět diagram aktivit synchronizace všech vláken. Synchronizace je prováděna pouze jedním vláknem, a to s $ID = 0$. Ve smyčce *while* je porovnáváno, zda byly nalezeny všechny matice **H**. Po úspěšném nalezení nastává konec smyčky a nastavení hodnoty *Hfound* na *true*.



Obrázek 6.10: Diagram aktivit pro synchronizaci všech vláken

Zdrojový kód synchronizace všech vláken lze vidět ve výpisu 6.9. Po skončení smyčky *while* je provedeno resetování hodnot, následuje nastavení hodnoty společné proměnné *Hfound* na true.

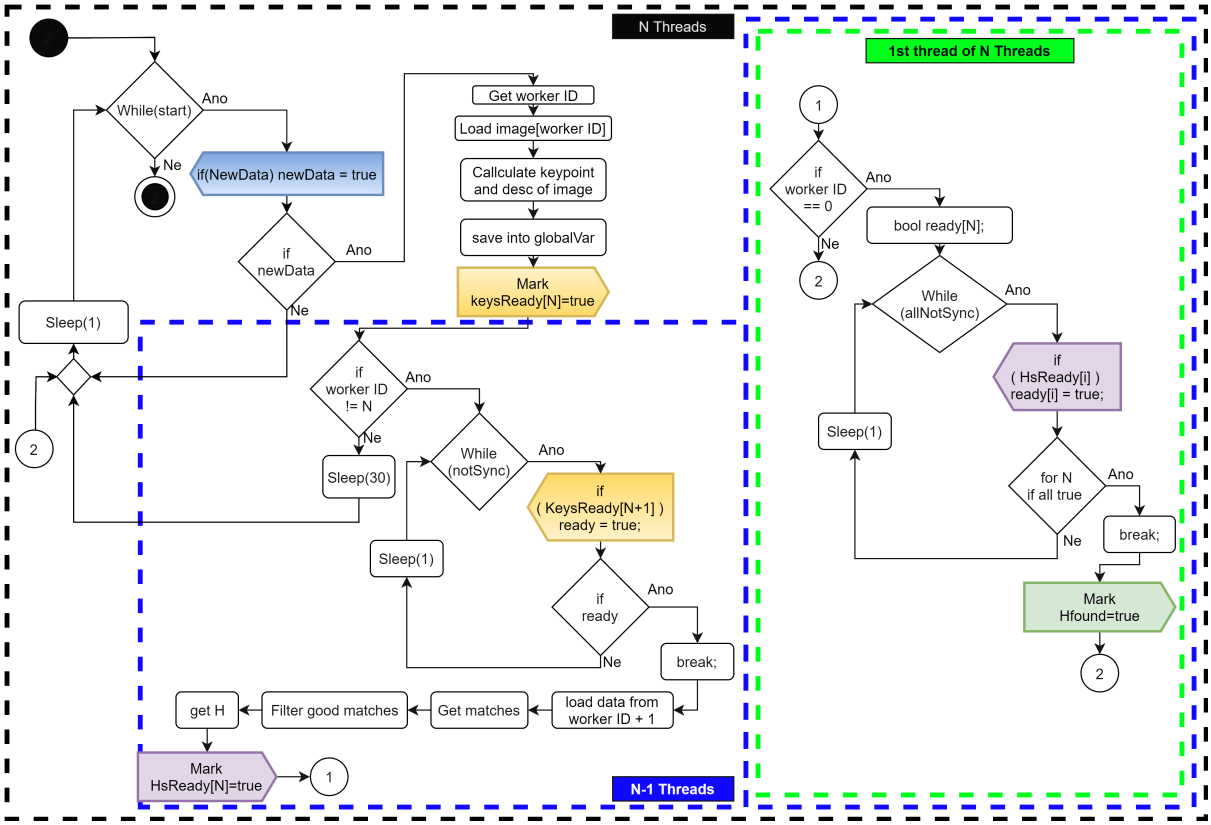
```
if (workerID == 0) {
   bool* backUp = new bool[Hsize];
   for (int i = 0; i < Hsize; i++) backUp[i] = false;
   bool allNotDone = true;
   while (allNotDone) {
      for (int i = 0; i < Hsize; i++) { // Nalézt změny
         if (!backUp[i]) {
            MHsReadyMutex[i].lock();
            backUp[i] = MHsReady[i];
            MHsReadyMutex[i].unlock();
      } }
```

```
        for (int i = 0; i < Hsize; i++) { // Zkontrovat výsledky
            if (!backUp[i]) break;
            else if (i >= (Hsize - 1)) allNotDone = false;
    } }
    // RESET HODNOT

    HfoundMutex.lock();
    Hfound = true;
    HfoundMutex.unlock();
}
```

Výpis 6.9: Zdrojový kód synchronizace všech vláken

Diagram aktivit procházených celků a společný pro všechna vlákna je zobrazen na obrázku 6.11. Černým přerušovaným ohraničením je znázorněna společná část pro všechna vlákna o počtu N. Modrým přerušovaným ohraničením je znázorněna část společná pro výpočet matic **H**, tedy počet vláken $N - 1$. Zelenou přerušovanou částí je ohraničeno první vlákno.

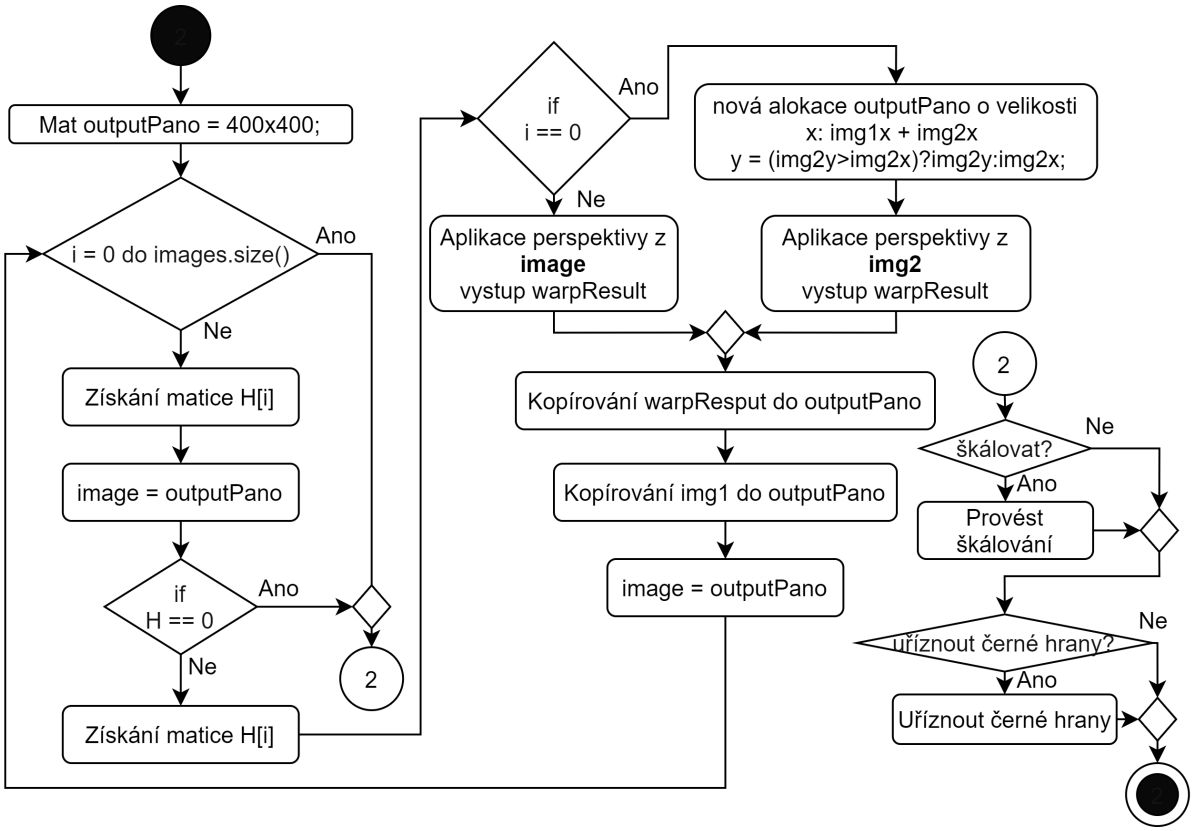


Obrázek 6.11: Spojené diagramy aktivit společné pro N vláken

6.2.9 Popis algoritmu pro spojování snímků

Pro vytvoření panoramatických snímků z vypočtených matic **H** a vstupních snímků byly vyvinuty dva způsoby, přičemž druhý způsob je rozšířením způsobu prvního. Spojení snímků se sestavuje ze dvou procesů, aplikování perspektivy pomocí matice **H** a spojení se sousedním snímkem. Pro aplikování perspektivy je využíváno funkce *warpPerspective()* vyžadující vstupní snímek, výstupní velikost, výstupní snímek a další volitelné parametry. Pro spojení sousedních snímků je zapotřebí základních operací s maticemi, kdy dojde ke kopírování snímku prvního do snímku s aplikovanou perspektivou. Tento základní kámen tvoří první způsob spojování, ten je rozšířen navíc o možnost spojit takto N snímků vedle sebe. Podmínkou však je, aby snímky byly sousední.

Pro zajištění spojování N snímků je vždy v nadcházejícím spojení nahrazen druhý snímek již vytvořeným panoramatem. Lze tak učinit při volání funkce *warpPerspective()*, kdy namísto snímku pro aplikování perspektivy je snímek nahrazen panoramatem. Aplikování perspektivy je tak provedeno na panorama z předešlého cyklu. Diagram aktivit algoritmu je zobrazen v obrázku 6.12. Algoritmus obsahuje kromě spojování snímků i ošetření kritických stavů. Ošetření vrácení nenulového snímku je již v prvním bloku, kdy je výstupní snímek *outputPano* alokován rozměrem 400×400 pixelů černé barvy. Pro případ, kdy z nějakého důvodu nemohla být matice **H** vypočtena, je její hodnota nulová. V případě, že se tento problém vyskytne, je algoritmus spojování okamžitě ukončen. Pro první cyklus je zapotřebí aplikovat perspektivu na snímek druhý, nikoliv na panorama, které ještě nebylo vytvořeno. Jako finální krok algoritmu je škálování výsledného panoramatu dle vstupních parametrů a následné oříznutí černých okrajů. Algoritmus se nachází v metodě názvem *getPanoramaImageT()*. Seřazení snímků v poli je z pravé strany do levé, algoritmus tedy spojuje postupně tímto směrem.



Obrázek 6.12: Diagram aktivit algoritmu pro spojování N snímků

Takto vytvořený panoramatický snímek však nemusí vždy při několikanásobném aplikování perspektivy vypadat dobře. Při velmi velkém zorném úhlu pořizovaných snímků dochází k značné deformaci prvních snímků. Ty mohou být natahovány až mimo rozsah výstupního snímku. Proto byl vyvinut druhý způsob spojování, který provádí přípravu před zavoláním metody *getPanoramaImageT()*, tedy prvního způsobu. Snímky jsou rozděleny na dvě části (dvě poloviny) a algoritmu je zaslána první polovina, která je spojovaná v orientaci směrem doleva. Poté je algoritmu zaslána druhá polovina, která je však zrcadlově transponovaná. Algoritmus opět spojuje snímky směrem doleva, ale snímek původně na konci řady je momentálně jako první a zrcadlený. Pro zrcadlené snímky však již vypočtené homografické matice **H** nesouhlasí, zrcadlení je tedy zapotřebí provést již po získání snímků a před výpočtem matic **H**. Na obrázku 6.13d a 6.13e lze vidět výsledný získaný panoramatický snímek pro levou a pravou polovinou.

(a) Levý snímek

(b) Prostřední snímek

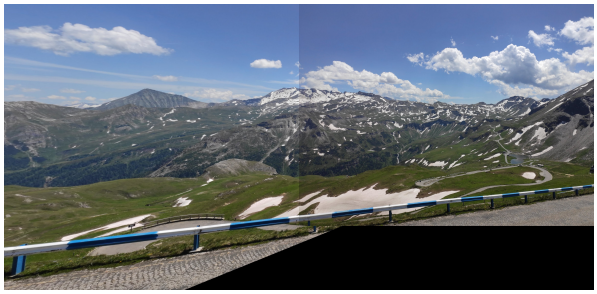
(c) Pravý snímek

(d) Vytvořené panorama z levého a prostředního snímku se zrcadlením

(e) Vytvořené panorama z prostředního a pravého snímku bez zrcadlením

Obrázek 6.13: Ukázka aplikace druhého způsobu spojování snímků

Pro spojení dvou panoramatů je nutno zrcadlené panorama opět zrcadlit. Lze si všimnout, že prostřední snímek je k sobě natočený (Obrázek 6.14a a 6.14b). Dalším krokem je oříznutí určité části prostředního snímku v obou panoramatech. Pro vyrovnanost bylo zvoleno oříznutí 50 % délky prostředního snímků v obou panoramatech. Oříznutí lze však aplikovat i poměrově, například oříznout 30 % v levém panoramatu a 70 % v pravém panoramatu. Po oříznutí obou panoramatů je zapotřebí krok spojení. Výsledkem je celkové panorama s efektem perspektivy na obou stranách, a tedy minimální deformace prvních snímků, jako to je při použití pouze prvního způsobu.



(a) Vytvořené oříznuté levé panorama

(b) Vytvořené oříznuté pravé panorama

(c) Výsledné spojené panorama

Obrázek 6.14: Vytvoření finálního panoramatu druhým způsobem spojování

Pro porovnání lze vidět na obrázku 6.15 stejné snímky zpracované pouze prvním způsobem. Prostřední snímek je roztažený přes celou pravou část panoramatu. Pravý snímek je mimo finální panorama. Celkově je prostřední snímek velmi deformovaný.

Obrázek 6.15: Panorama získané pouze použitím prvního způsobu

Jak již bylo řečeno v odstavcích výše, algoritmus druhého způsobu spojování snímků provádí pouze předpřípravu a závěrečné spojení, zdrojový kód lze vidět ve výpisu 6.10. Pro předávání snímků do metody *stitchMultiImages()* je využito páru vektorů snímků, dále pak vektoru matic **H**. Počet vektorů je stanoven polovinou počtu snímků metodou *getHalfIndex()*.

```
bool heStitcher::getPanorama() {
    std::vector<std::pair<Mat, Mat>> images;
    vector<Mat> Hs;
    Mat leftPanorama, rightPanorama;
    for (int i = 0; i < getHalfIndex(); i++) { // Získání snímků pro pravé panorama
        images.push_back(images_toCalc.at(i));
        Hs.push_back(this->Hs.at(i));
    }
    stitchMultiImages(images, Hs, rightPanorama, 500); // spojování první metodou
    this->outputImage = rightPanorama;
    images.clear(); // Reset vektoru obrázku
    Hs.clear(); // Reset vektoru matic
    for (int i = getHalfIndex(); i < this->Hs.size(); i++) { // získání snímků pro levé panorama
        Mat image1 = this->images_toCalc.at(i).first.clone();
        Mat image2 = this->images_toCalc.at(i).second.clone();
        flip (image1, image1, 1);
        flip (image2, image2, 1);
        images.push_back(std::make_pair(image1, image2));
        Hs.push_back(this->Hs.at(i));
    }
    stitchMultiImages(images, Hs, leftPanorama, 500); // spojování první metodou
    flip (leftPanorama, leftPanorama, 1); // Zrcadlení levého panoramatu
    int width = images.at(this->Hsize).cols * this->outputResize;
    cv::Rect myROI1(0, 0, leftPanorama.cols - width / 2.0, leftPanorama.rows);
    cv::Rect myROI2(width / 2.0, 0, rightPanorama.cols - width / 2.0, rightPanorama.rows);
    Mat image1 = leftPanorama(myROI1).clone(); // Oříznutí levého panoramatu
    Mat image2 = rightPanorama(myROI2).clone(); // Oříznutí levého panoramatu
    cv::hconcat(image1, image2, this->outputImage); // Spojení dvou panoramatů
    return true;
}
```

Výpis 6.10: Zdrojový kód druhého způsobu spojování snímků

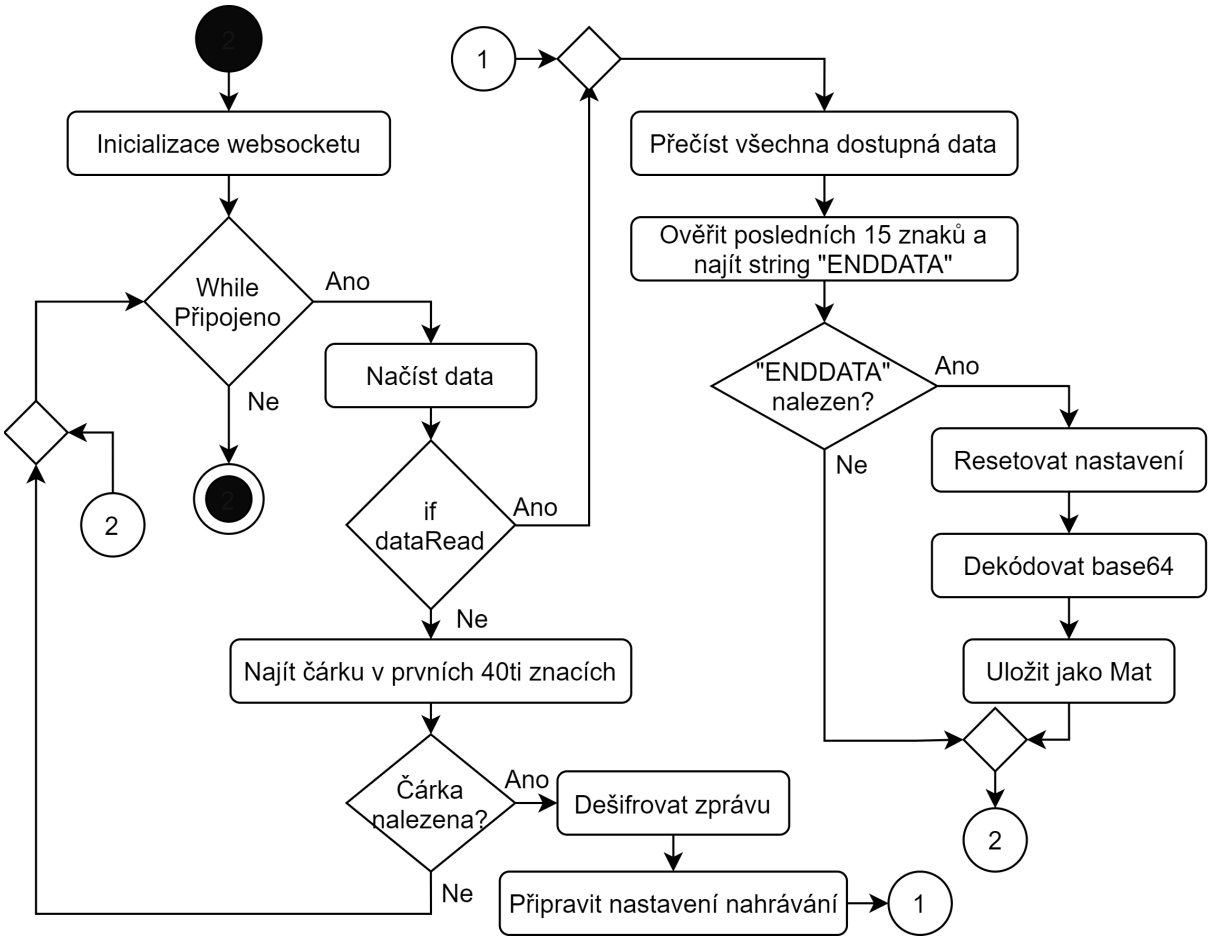
6.2.10 Komunikace pomocí *websocketu*

Po spuštění aplikace je spuštěn HTTP server obsahující podporu *websocketu*. Pro komunikaci je využíváno kódování *base64*, které je komprimovaným tvarem dat a reprezentováno jako text. Toto kódování je velmi populární pro webové prohlížeče i služby. V důsledku velkého objemu dat jsou zasílaná data rozdělována na několik paketů. K rozlišení začátku a konce snímků byl přidán na začátek zprávy text označující čárkou začátek dat snímku. Pro rozpoznání, že se jedná o kamerový

snímek, je přidán text „CAM". Pro rozpoznání levého, pravého či prostředního snímku je přidávaná předložka „L", „M" nebo „R". Konec zprávy je definován textem „ENDDATA". Objem dat může dosahovat až 1 kB, záleží na kompresním poměru a velikosti snímku. Pro FullHD snímek s kompresním faktorem 0,9 je velikost přibližně 200 kB. Velikost objemu dat jednoho paketu lze nastavit na straně klienta, pokud nastavená není, je zvolena prohlížečem maximální povolená velikost v závislosti na používaném API *websocketu*. Pro webový prohlížeč *Chrome* je tato maximální velikost paketu 131 kB. Formát zprávy lze vidět na obrázku 6.16.

Obrázek 6.16: Ukázka formátu zprávy skrze *websocket*

Pro zpracování zprávy je sestaven příslušný dešifrovací algoritmus, jehož diagram aktivit je zobrazený na obrázku 6.17. Po vytvoření požadavku pro otevření *websocketu* ze strany klienta je provedena inicializace. Následuje smyčka *while*, která končí až po ukončení připojení ze strany klienta či serveru. Ve smyčce jsou přečtená data uložena do pole hodnot datového typu *char*. V dalších krocích se již využívá datového typu *string*. Je tak dáno knihovnou *POCO*, která neobsahuje definice metod pro práci s datovým typem *string*. Prvním krokem pro zahájení čtení dat snímků je zapotřebí detekovat přítomnost začátku zprávy nalezením čárky. Hodnota *dataRead* je v tento okamžik nastavena na *false*. V případě, že byla čárka nalezena, provede se dešifrování řetězce před čárkou. Konfigurace je přečtena z řetězce a na jejím základě provedeno nastavení nahrávání. Hodnota *dataRead* je nastavena na hodnotu *true*. Data snímků jsou cyklicky čtena. Na každém konci současného paketu je provedena detekce přítomnosti řetězce „ENDDATA". Jakmile je řetězec nalezen, je nastavení resetováno a provedeno dekódování z formátu *base64* do formátu *JPEG*, poté je proveden převod na datový typ *Mat* využívající knihovnou *OpenCV*.

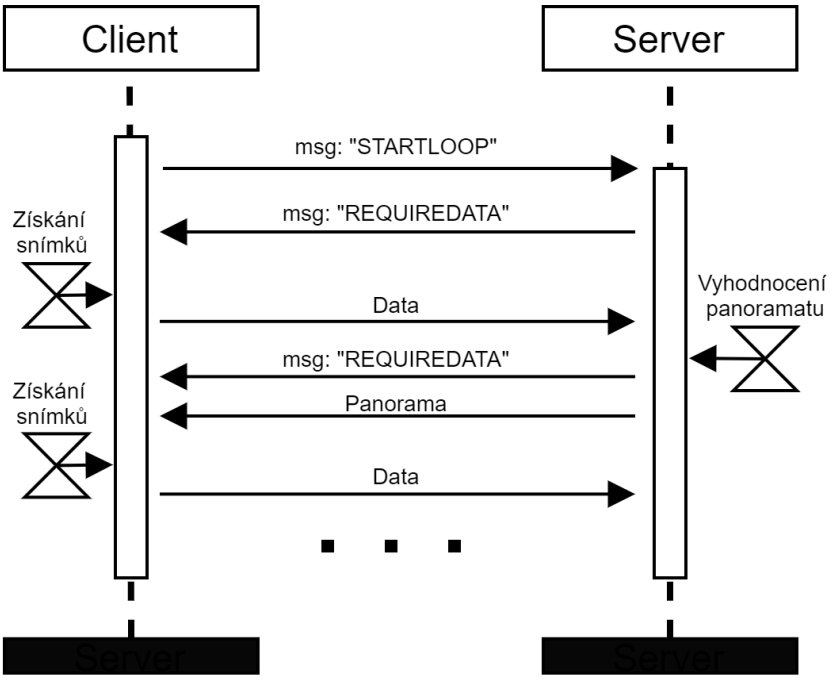


Obrázek 6.17: Diagram aktivit algoritmu pro zpracování zprávy skrze *websocket*

Algoritmus neobsahuje ošetření stavu, kdy by ukončovací řetězec byl skrze dva pakety. V takovémto případě můžou nastat dvě možnosti. Dojde k chybě zpracování snímku. Snímek bude obsahovat data následujících snímků včetně prvotního konfiguračního řetězce. Bude špatně dekódován a přeložen. Takováto deformace se projeví nulovou hodnotou matice snímku, která je dále ošetřena při čtení vstupních snímků ve třídě *trStitcher*. Nedojde tedy k pádu aplikace a budou vyžádány nové snímky. Druhou možností je vyskytnutí chyby při posledním zasílaném snímku. Ten nikdy nenarazí na ukončovací řetězec, a tudíž nikdy nedojde k ukončení čtení. Smyčka *while* bude pozastavena v metodě pro čtení dat, která již nepřichází.

Sekvence příchozích a odchozích zpráv je znázorněna na obrázku 6.18. Při vytvoření spojení je odeslána inicializační zpráva „STARTLOOP". Na základě tohoto tvaru je na straně serveru provedena inicializace smyčky *while* a její odstartování. V rámci inicializace je rovněž vrácena odpověď pro získání dat ve tvaru „REQUIREDATA". Pokud webový klient zaznamená takovouto příchozí zprávu, připraví aktuální snímky a zašle sekvenci snímků v definovaném formátu serveru. Po přijmutí snímků serverem a získání panoramatu je zažádáno nejdříve o další data a posléze zasláno

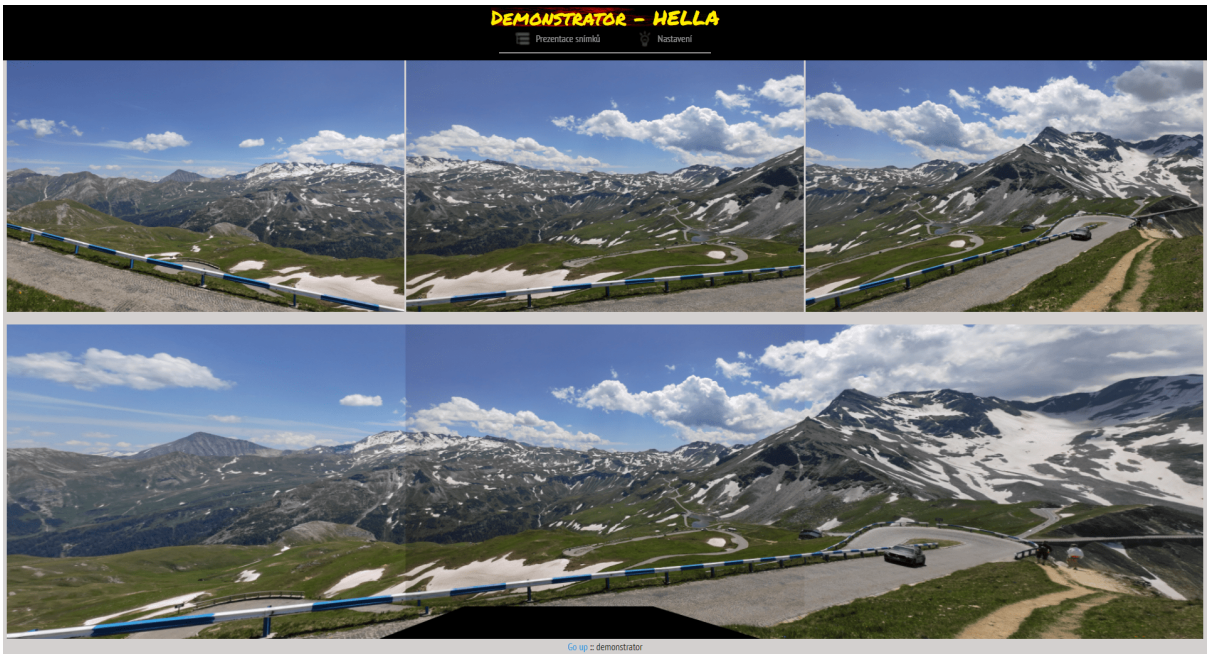
panorama. Takto je vykonáváno cyklicky do ukončení *websocketu* klientem nebo serverem.



Obrázek 6.18: Sekvenční diagram zpráv skrze *websocket* jedné smyčky

6.3 Webové rozhraní

Webové rozhraní je tvořeno třemi vstupními snímky a výstupním panoramatem. Obsahuje sekci pro zobrazení obrázků a sekci pro nastavení. Pro správné fungování stránky je zapotřebí zakázat kontrolování zdrojů webových elementů neboli politiku *CORS* ve webovém prohlížeči. Webové rozhraní disponuje zobrazením živého přenosu ze tří kamer, odesíláním snímků aplikaci pomocí *websocketu*, správou *websocketu* a možnost přepínat mezi živým přenosem a lokálními demo snímky. Výhodou webového rozhraní je možnost zasílat jakékoliv snímky prostřednictvím *websocketu* aplikaci, a tak jednoduše testovat algoritmus pro spojování snímků. Zdrojové kódy jsou tvořeny jedním *HTML* souborem, několika *CSS* soubory a několika *Javascript* soubory. Ke stylování webu je využito nástroje *Boostrap*. *Javascript* využívá knihovny *Jquery*.



Obrázek 6.19: Vizuální podoba webového rozhraní

Pro spuštění webového prohlížeče bez politiky *CORS* je zapotřebí deaktivace. Pro webový prohlížeč *Chrome* lze tak učinit spouštěcími parametry aplikace. Od verze 86 je však zapotřebí kromě příkazu k deaktivaci uvést i cestu k profilu uživatele. Pro prohlížeč *Firefox* je deaktivace prováděna v konfiguračním souboru, který lze upravovat i během chodu aplikace. Názorná ukázka deaktivace je popsaná ve výpisu 6.11.

```
// Chrome spouštění parametry
Target:
"C:\Program Files\Google\Chrome\Application\chrome.exe" --disable-web-security --disable-site-isolation-
    trials ----disable-web-security --user-data-dir="C:\src\user"

// --- Firefox --- //
// do www vyhledávání zadat:
about:config
// najít uvedený řádek:
security.fileuri.strict_origin_policy = false
```

Výpis 6.11: Deaktivace politiky CORS

Pro přijímání živých snímků je vytvořena třída psaná v *Javascriptu* skládající se z funkcí pro obnovování snímků, z funkce pro případ chyby obnovení a z funkce pro započetí (výpis 6.12). Tyto funkce jsou přiřazeny elementu *img* s definovaným *ID* pro každou kameru zvlášť. Pro obnovování snímků je využita funkce *refreshImage()*. Jakmile je snímek obnoven, je zavolaná funkce *onImageLoaded()*, která zajistí obnovovací frekvenci definovanou při spuštění a opět zavolá funkci pro obnovení.

V případě chyby je zavolaná funkce *onImageError()*, která se pokusí změnit URL adresu z formátu *MJPEG* na formát *JPEG* a provést vyčtení po specifikované době. Pro obnovení snímku prohlížečem je zapotřebí snímkům přiřazovat čas, aby prohlížeč vyhodnotil, že se jedná o změnu snímků. Pokud by takto nebylo provedeno, obnovování snímků by nebylo vizuálně prováděno prohlížečem.

```
function refreshImage( ) {
        if ( mjpegMode ) {
            imageElement.src = mjpegUrl;
        } else {
            timeStart = new Date( ).getTime( );
            imageElement.src = jpegUrl + '?t=' + timeStart;
    }   }
    function onImageError( ) {
        imageElement.src = "images/noCamera-min.jpg";
        mjpegMode = !mjpegMode;
        setTimeout( refreshImage, 1000 );
    }
    function onImageLoaded( ) {
        if ( !mjpegMode ) {
            var timeTaken = new Date( ).getTime( ) - timeStart;
            setTimeout( refreshImage, ( timeTaken > frameInterval ) ? 0 : frameInterval - timeTaken );
    }   }
```

Výpis 6.12: Zdrojový kód pro obnovování snímků z kamer

Zajištění konkrétního zdroje je zprostředkováno funkcí *start()*, která vyžaduje tří parametry, tedy FPS, URL adresu zdroje a *ID* elementu. V této funkci je provedena inicializace a přiřazení funkcí elementu s daným *ID*. Ve výpisu 6.13 je znázorněna definice funkce *start()*, její zavolání a vytvoření třídy.

```
var start = function( fps, url , cid ) {
    jpegUrl = url + '/camera/jpeg';
    mjpegUrl = url + '/camera/mjpeg';
    imageElement = document.getElementById( cid );
    imageElement.onload = onImageLoaded;
    imageElement.onerror = onImageError;
    mjpegMode = true;
    refreshImage( );
}

// vytvoření třídy a započetí obnovování
camL = Camera;
camL.Start( 30 , 'http://192.168.1.150:8000', 'cameraL' );
```

Výpis 6.13: Zdrojový kód pro obnovování snímků z kamer

Pro vytvoření spojení mezi klientem a serverem je využito nástroje *websocket*, jenž je součástí řady prohlížečů. Ve funkci *createWebsocket()* je rovněž ověřovaná tato skutečnost (výpis 6.14). Pro případ, kdy nástroj není součástí prohlížeče, je uživatel na tuto skutečnost upozorněn a navázání není provedeno. Pro navázání spojení je zapotřebí znát IP adresu a PORT serveru. Po vytvoření spojení je odeslána první zpráva, poté jsou sledovány příchozí zprávy v návratové funkci *onmessage*. V návratové funkci je přijímána zpětná vazba serveru, která je rozpoznána a vykonán požadavek.

```
var ws;
function createWebsocket(IP,PORT) {
    if ("WebSocket" in window) {
        //ws = new WebSocket("ws://127.0.0.1:9980/ws");
        ws = new WebSocket("ws://"+IP+":"+PORT+"/ws");
        ws.onopen = function() {
            ws.send("STARTLOOP");
        };
        ws.onmessage = function(evt) {
            var text = evt.data + "";
            text = text.trim();
            if(text.includes("REQUIREDATA")) {
                sendImages();
            }
            if(text.includes("PANORAMA")) {
                . . .
            }
        };
        ws.onclose = function() { . . . };
    else {
        alert("This browser does not support WebSockets.");
    }
}
```

Výpis 6.14: Vytvoření *websocketu* pomocí *Javascriptu*

Pro vyjádření principu získávání snímků z webového rozhraní a zasílání skrze *websocket* jsou ve výpisu 6.15 znázorněny zjednodušené funkce. Po zavolání funkce *sendImages()* z funkce *websocketu* je vyžádáno obnovení snímků z elementu *img* do elementu *canvas*. Poté je snímek z elementu *canvas* přeložen do formátu *base64* prostřednictvím funkce *toDataURL()*. Tato funkce vrátí formát dat současně se začátkem řetězce „" následujícím řetězcem dat snímku. Před řetězec je přidán další řetězec rozhodující o pořadí kamery (graficky znázorněno na obrázku 6.16).

```
function drawContext(elementID, canvasID) {
        var v = document.getElementById(elementID);
        v.crossOrigin = "Anonymous";
        var canvas = document.getElementById(canvasID);
        var context = canvas.getContext('2d');
        var cw = canvas.width; var ch = canvas.height;
        context.drawImage(v,0,0,cw,ch);
}

function getDataFromC(canvasID, prefix) {
    var sourceCanvas = document.getElementById(canvasID);
    var dataURL = sourceCanvas.toDataURL('image/jpeg', 0.9);
    return data = prefix + dataURL + "ENDDATA";
}

function sendImages() {
    drawContext('cameraL', 'cL');
    var data = getDataFromC('cL', "LCAM");
    ws.send(data);
}
```

Výpis 6.15: Získání snímků z webového rozhraní a odeslání dat

Z nastíněného řešení je zřejmé jednoduché přepínání snímků kamer na snímky libovolné. Je tak provedeno pouhou záměnou zdroje obrázků v elementu *img*. Pro změnu snímků uživatelem bez zásahu do zdrojového kódu HTML souboru jsou k dispozici demonstrační snímky již součástí webu, které jsou změněny po kliknutí na tlačítka v okně nastavení.

Kapitola 7

Testování vytvořeného systému

Pro otestování vytvořeného systému byly provedeny testy jednotlivých aplikací, tedy odzkoušení živého přenosu z kamer a vyhodnocování panoramatu. Cílem testů bylo ověřit dlouhodobou stabilitu systému a kvalitu poskytování služeb.

7.1 Testování panoramatických snímků

Pro správné vyhodnocování panoramatických snímků je zapotřebí volby vhodného algoritmu pro získání klíčových bodů a deskriptorů, avšak mnohem důležitější je provedení filtrace shod, které je v poskytnutém řešení zajištěno pouze prostřednictvím porovnání vzdáleností s možností volby procentuálního poměru vzájemných vzdáleností. Pro větší filtraci shod je volen menší procentuální poměr, kdy jsou vylučovány vzdálenější shody. Může však docházet i k nalezení blízkých shod, které byly přiřazeny nesprávně. Cílem provedeného testování je posouzení výsledného panoramatu na základě statistického souboru hodnot. Z kapitoly páté je zřejmý princip spojování dvou snímků a využití homografické matice. Dva snímky takto spojovány do sebe se navzájem protínají (Obrázek 4.7b). Lze tedy tvrdit, že dva snímky obsahují stejnou sekvenci hodnot pixelů ve vertikálním směru v určitém místě. Po aplikování perspektivy na snímek druhý, na základě vypočtené homografické matice, je získán snímek s posunutím právě do této společné oblasti. Získáním souboru hodnot pixelů právě v této oblasti lze porovnávat pixely z obou snímků a vyhodnocovat správnost spojení.

V rámci testování je nejdříve provedeno vizuální posouzení generovaných panoramatických snímků a roztřídění do 4 kategorií, tedy výborný, přijatelný, nedostatečný a neúspěšný. Kategorie výborné vyhodnocení splňují snímky, kde nedochází k žádné nebo minimální neshodě na úrovni přechodů snímků. Přijatelným vyhodnocením snímků jsou označeny všechny snímky, kdy přechod mezi snímky je z větší části vyhodnocen správně. Pro zkreslené snímky efektem rybího oka je posuzována horizontální rovina středu panoramatu. Pro lineární snímky je posuzován celkový přechod. Jako nedostatečná panoramata jsou označeny všechny ostatní snímky, kde nedošlo ke ztrátě informace, ale přechod je fázově posunut. Všechna panoramata značně deformovaná či neobsahující

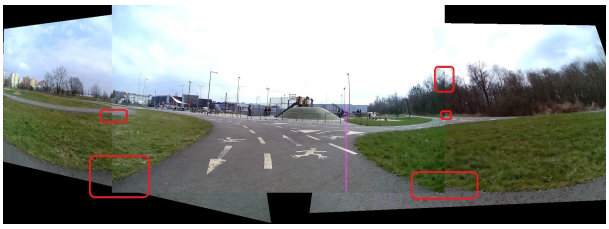
všechny tři vstupní snímky jsou značena jako neúspěšná. Pro každý panoramatický snímek je vyhodnocen medián hodnot a RMS ze získaného souboru pixelů, které jsou dále rozlišeny pro spojení levé a pravé strany panoramatu. Filtrační poměr vyhodnocených shod je stanoven pro celé testování na hodnotu 65 %. Tato hodnota byla určena na základě předešlých testů, kdy bylo zjištěno, že vyšší filtrační poměr (tedy nižší hodnota) má za následek filtrování i správných shod, které jsou nezbytné pro správné vyhodnocení. Nižší filtrační poměr (tedy vyšší hodnota) zase ponechává nesprávné shody. Zvolenou metodou pro získání klíčových bodů a deskriptorů je metoda *SIFT*. Vstupní škálovací poměr snímků byl nastaven na 50 %. Snímky jsou vyhodnocovány černobíle.

Celkem bylo vyhodnoceno 128 panoramatů tvořených ze tří snímků. 282 snímků (93 panoramatů) bylo pořízeno kamerovým modulem obsahujícím čočku se zkreslením rybího oka. Zbylých 84 snímků (28 panoramatů) bylo vyhodnoceno lineární čočkou mobilního telefonu. Zkreslené snímky byly vyhodnoceny a rozděleny do kategorií o počtech 10 výborných, 26 přijatelných, 45 nedostatečných, 11 neúspěšných a 6 algoritmem nevypočtených. Pro lineární snímky bylo rozdělení o počtech 23 výborných, 4 přijatelných, žádných nedostatečných a 2 neúspěšných. Příklad panoramatických snímků různých kategorií je vidět na obrázku 7.1, kde vady přechodu mezi snímky jsou značeny červeným označením.

Po získání 4 souborů hodnot pixelů a převedením do černobílého rozsahu 0 až 255 je provedeno porovnání dvou pixelů se shodnou lokací pro pravou a levou stranu panoramatu. Porovnání je provedeno jako absolutní rozdíl hodnot pixelů. Ze získaných rozdílů hodnot je následně vypočten medián a RMS. Snímky však mohou mít různé úrovně jasu, pro korekci jasu je aplikováno posunutí hodnot pixelů o definovanou hodnotu. Hodnoty pixelů jsou generovány automaticky zdrojovým kódem do textového souboru. Vykreslení do grafů a aplikování posunutí je provedeno prostřednictvím programu *Office 365 Excel*. Na obrázku 7.2 je uveden příklad vyhodnocení hodnot. Snímek je na první pohled vyhodnocen perfektně, mezi snímky se však nachází značné rozdíly jasu. Dále na spodních částech přechodů mezi snímky na obou stranách panoramatu lze vidět nekorektní přechod, avšak minimální. Tato chyba je rovněž vidět ve grafech na obrázku 7.2b a 7.2c.

Získané hodnoty panoramatu z obrázku 7.2a bez aplikace vyvážení jasu vytváří pro levou stranu medián o hodnotě 12 a RMS o hodnotě 21,67. Pro pravou stranu je medián roven 44 a RMS rovno 47,799. Tyto hodnoty však jsou předpokládané v důsledku rozdílu jasu. Po jeho vyvážení je získány pro levou stranu medián roven 11 a RMS rovno 17,856 a pro pravou stranu medián roven 18 a RMS rovno 25,696.

(a) Lineární vstupní snímky, přijatelný



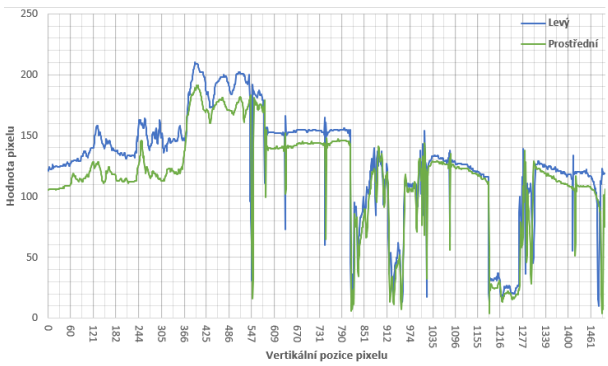
(b) Zkreslené vstupní snímky, přijatelný

(c) Zkreslené vstupní snímky, nedostatečný

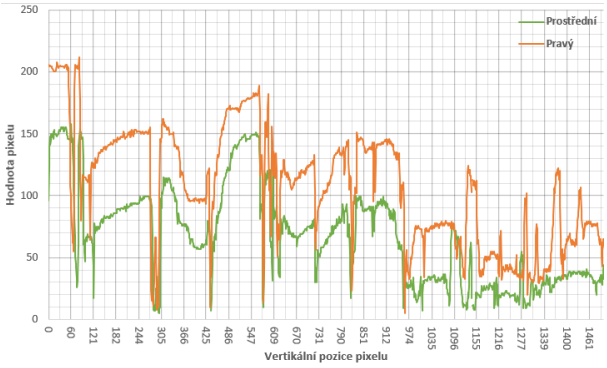
(d) Zkreslené vstupní snímky, výborný

Obrázek 7.1: Příklady rozdělení snímků do kategorií na základě celkové kvality vyhodnocení

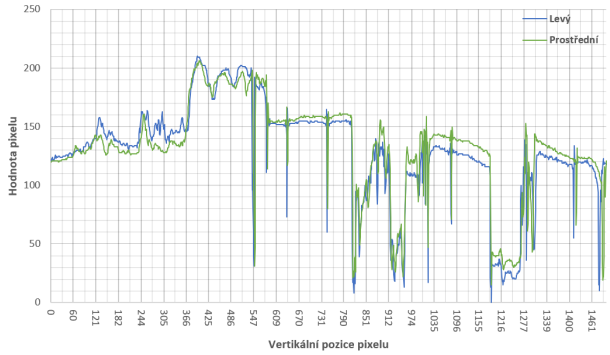
(a) Vstupní panoramatický snímek - univerzitní aula *VŠB-TUO*



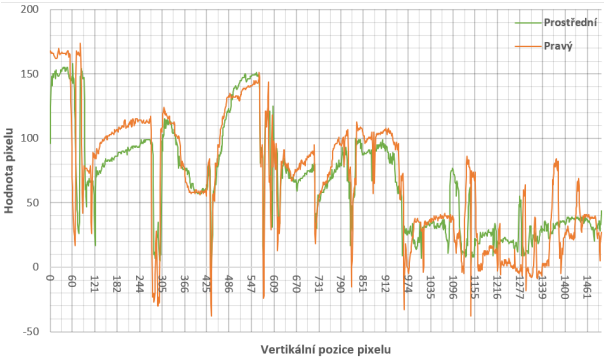
(b) Vykreslení hodnot, levá strana



(c) Vykreslení hodnot, pravá strana



(d) vyvážení jasu, levá strana

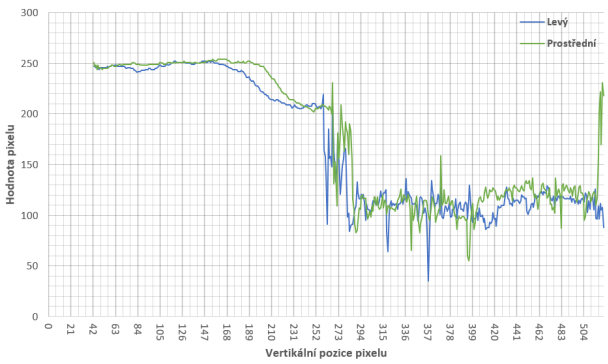


(e) vyvážení jasu, pravá strana

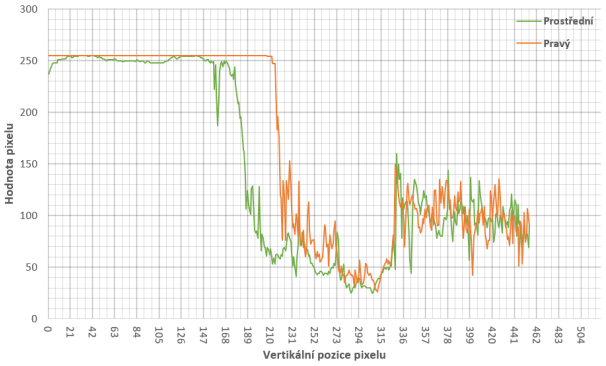
Obrázek 7.2: Vykreslení hodnot pixelů pro levou a pravou část panoramatu a vyvážení jasu

Obdobnému způsobu vyhodnocováním bylo podrobeno všech 33 panoramatických snímků s vizuálním hodnocením výborný. Bylo zjištěno, že se hodnoty mediánu pohybovaly pod hodnotu 19 a hodnoty RMS pod hodnotu 26,792. Při testování panoramatických snímků s hodnocením přijatelné odpovídala hodnota RMS ve většině případů vyšším číslům než 26,792. Vyskytly se však i případy, kdy ve snímcích bylo méně různorodých ploch a přechodů. To mělo za důsledek, že i v případě možné nekorekce byl pixel stejného odstínu. Dalším faktorem je převod RGB složek na jednotný odstín, kdy modrá a zelená složka při převodu nemají takové váhy, jako červená. Celkové vyhodnocení takového panoramatu bylo pak podle hodnoty RMS jako výborný. Příkladem chybného vyhodnocení je panoramatický snímek na obrázku 7.3a. Hodnota RMS levé strany panoramatu je 21,726 a hodnota RMS pravé strany pak 49,239. Porovnávání pixelů pro kategorii nedostatečné a neúspěšné je již podle hodnoty RMS dostačující. Příklady dalších vyhodnocení lze nalézt v příloze B, C, D a E této práce.

(a) Vstupní panoramatický snímek č. 67 s hodnocením přijatelný



(b) Vykreslení hodnot, levá strana

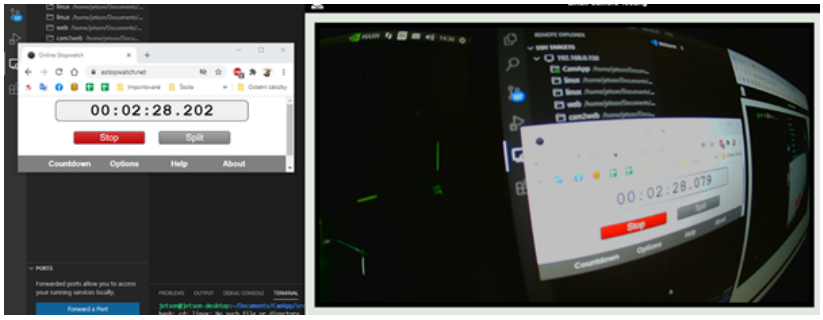


(c) Vykreslení hodnot, pravá strana

Obrázek 7.3: Vykreslení hodnot pixelů pro levou a pravou část panoramatu č. 67

7.2 Testování živého přenosu z kamer

Pro otestování aplikace poskytující živý přenos z kamery prostřednictvím webového serveru byla měřena doba odezvy snímaného obrazu a doručení na obrazovku klienta a provedena zkouška dlouhodobého provozu. Měření odezvy vysílaného obrazu bylo provedeno spuštěním časovače obsahujícího sekundy a mikrosekundy. Následně byla kamera namířena na monitor počítače, aby snímala hodnotu časovače. Pro získání odezvy bylo zapotřebí získat hodnotu časovače a snímek kamery v jeden okamžik. Toho bylo docíleno využitím funkce *Windows* pro vytisknutí aktuálního snímku obrazovky. Snímek je vyobrazen na obrázku 7.4. Data obrazu kamery pořízené na platformě *Jetson Nano*, odeslané kabelovým připojením skrze síť ethernet a skrze jeden *router* do osobního počítače s otevřeným webovým rozhraním trvalo přibližně 123 ms ±5 ms. Pro zkoušku dlouhodobého provozu bylo živé vysílání spuštěno po dobu jedné hodiny. Poté byla stejným způsobem sejmuta odezva, která byla stále shodná.



Obrázek 7.4: Měření odezvy vysílaného snímku

Závěr

Práce byla vypracována v rámci *trainee* programu ve společnosti *Hella*. Výsledky práce budou základem pro další vývoj platformy předního nárazníku *SmartFace*. V rámci diplomové práce je vytvořen obslužný software a stojan reprezentující přední nárazník pro pobočku v *Ostravě*. V dalším vývoji lze zdrojový kód převést do jazyka popisující hardware (VHDL, verilog nebo jiných), následně do hradlového formátu FPGA a při masovém nasazení do integrovaných obvodů ASIC.

Obslužný software sestává ze tří dílčích programů na sobě nezávisle běžících, tedy z aplikace pro zpracování snímků z kamer, aplikace pro zpracování panoramatických snímků a webového rozhraní. Vytvořen je rovněž stojan s boxem reprezentující rozmístění senzorů v chytrém nárazníku. Vytvořená Obslužný software slouží pro demonstrační účely a jako stavební kámen budoucích projektů.

Aplikace pro zpracování snímků z kamer je snadno přenositelná a lze ji spustit pod operačním systémem *Windows* i *Linux*. Bylo docíleno poskytování rozlišení 1920×1080 při 30ti snímcích za vteřinu až pro tři kamery najednou. Využitým médiem pro sběr dat z kamer je sběrnice USB.

Aplikace pro zpracování panoramatických snímků je schopná vyhodnocovat až pro N vstupních snímků. Vyhodnocení pro tři FullHD snímky se vstupním polovičním škálováním trvá přibližně $80\,\text{ms} \pm 40\,\text{ms}$, kdy záleží na počtu klíčových bodů a vyhodnocených shod, vyhodnocováno je pouze pomocí CPU. Lze volit z několika parametrů, jakými jsou například volba vstupního a výstupního škálování, vyhodnocování v odstínech šedi, volba počtu snímků k vyhodnocení, volba filtračního faktoru, volba metody pro detekci bodů a další. Nabyté znalosti zpracování obrazu pomocí knihoven *OpenCV* jsou přehledně k dispozici ve zdrojových kódech aplikace a snadno upravitelné ve stávajícím řešení. Přenositelnost aplikace pro výpočet panoramatických snímků bohužel není již tak snadná. Je zapotřebí přítomnosti knihoven *OpenCV* a *POCO*, jejichž instalace může být v některých případech komplikovaná. Aplikace je psaná pro systém *Windows*.

Webové rozhraní se sestavuje z prezentování živého přenosu tří kamer a vyhodnocovaného panoramatického snímku. Disponuje komunikací pomocí *websocketu* a umožňuje přepínat mezi snímky z kamer a snímky demonstračními.

Algoritmy pro detekci klíčových bodů a deskriptorů (nabízené knihovnami *OpenCV*) jsou *SIFT*, *SURF*, *FAST* a *OBR*. Nejlepší výsledky, jichž je možné použít bez filtrace k vyhodnocení, poskytuje metoda *ORB*. Nicméně výsledků je mnohdy málo a nestačí k dalšímu zpracování. Druhou nejvhod-

nější metodou je algoritmus *SIFT*, který dodává potřebné množství informací bez velkých odchylek. Metody *SURF* a *FAST* jsou zase velmi rychlé algoritmy, nicméně s velkou chybovostí.

Pro demonstraci rozmístění senzorických prvků v předním chytrém nárazníku *SmartFace* byl vytvořen snadně manipulovatelný stojan z hliníkových profilů a s volitelným rozpětím. Pro umístění výpočetní techniky slouží box, rovněž vytvořen z hliníkových profilů. Konstrukce jsou robustní, nicméně úskalím podstavy je, že při rozechvění nárazem je potřebná dlouhá doba ke stabilizaci. Stojan obsahuje navržený elektronický subsystém sestávající se z kamer, kabeláže, výpočetní techniky a napájecí soustavy. Očekávané rozšíření o další senzorické či výpočetní části je v budoucnu možné. V rámci testování bylo docíleno zpětné kontroly vyhodnocených panoramatických snímků a určení hranice hodnoty RMS, na základě které lze určit, zda je snímek vyhodnocen výborně, přijatelně, nedostatečně či neúspěšně. Rovněž byly provedeny výkonové dlouhodobé testy pro kontrolu správného fungování i po několikanásobném vyhodnocení, jež skončily pozitivně.

Naprogramovaný obslužný software nachází budoucí uplatnění pro případ rozšíření o další funkcionality či přepis pro operační systém *Linux*, neboť aplikace je svou strukturou přehledná a využívá minimum funkcí operačního systému *Windows*. Možným rozšířením funkcionalit je například implementace zpracování obrazu pomocí grafické akcelerace skrze nástroj *CUDA* nebo *OpenCL*. Dále je možná implementace konfigurace skrze konfigurační soubor, možnost ovládání algoritmu skrze webové rozhraní, přidání možnosti konfigurovat vstupní snímky a upravovat například jas, sytost a podobně, a nebo implementovat rozpoznávání správnosti panoramatu na základě hodnoty RMS, což bylo zjištěno v rámci testování.

Literatura

1. *OpenCV geometrická transformace dokumentace. Open Source Computer Vision [online]. 2020: Intel Corporation, Willow Garage, Itseez, 2020 [cit. 2021-03-17].* Dostupné také z: https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga7dfb72c9cf9780a347fbe3d1c47e5d5a.
2. *Microcontrollers for the Automobile. Https://www.mcjournal.com/ [online]. Micro Control Journal, 2009 [cit. 2020-10-05].* Dostupné také z: https://www.mcjournal.com/articles/arc105/arc105.htm.
3. *DONOVAN, John. What Engineers Need to Know When Selecting an Automotive-Qualified MCU for Vehicle Applications. Digi Key electronics [online]. 701 Brooks Avenue South, Thief River Falls, MN 56701 USA: Digi-Key Electronics, 2014 [cit. 2020-10-05].* Dostupné také z: https://www.digikey.com/en/articles/what-engineers-need-to-know-when-selecting-an-automotive-qualified-mcu-for-vehicle-applications.
4. *JURGEN, Ronald K. Automotive Microcontrollers. SAE International, 2008. ISBN 978-0768020670.*
5. *SALEEM, Waqar. Ten things to look for when choosing MCUs for automotive designs. Embedded.com [online]. 245 Main Street Cambridge, MA 02142 USA: Aspencore, 2012, 3. prosince 2012 [cit. 2020-10-05].* Dostupné také z: https://www.embedded.com/ten-things-to-look-for-when-choosing-mcus-for-automotive-designs/.
6. *ISO 26262. 2nd edition. Švýcarsko: International Organization for Standardization, 2018.*
7. *Mgr. ŘEHOŘOVÁ, Lucie. Označení CE – přeceňovaná zkratka. Retail News [online]. Praha: Advokátní kancelář REHAK LEGAL, 2015 [cit. 2020-11-17].* Dostupné také z: https://retailnews.cz/2015/04/02/oznaceni-ce-precenovana-zkratka/.
8. *Different Types of Microcontrollers are used in Automobile Applications. EL-PRO-CUS [online]. Indie - Hyderabad: ElProCus Technologies Pvt, 2020 [cit. 2020-11-17].* Dostupné také z: https://www.elprocus.com/different-microcontrollers-used-in-automobiles/.

9. *Infineon Technologies AURIX TC26xL 32-bit TriCore Microcontrollers. Mouser Electronics [online]. Brno: Mouser Electronics, 2020 [cit. 2020-11-17].* Dostupné také z: https://cz.mouser.com/new/infineon/infineon-aurix-tc26xl-tricore-mcu/.

10. *Obrázek MCU TriCore Aurix. In: Mouser Electronics [online]. Brno: Mouser Electronics, 2020 [cit. 2020-11-17].* Dostupné také z: https://cz.mouser.com/images/marketingid/2018/img/195635300.png?v=052020.0840.

11. *AVR MCUs. Microchip [online]. USA, Arizona: Microchip, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.microchip.com/design-centers/8-bit/avr-mcus.

12. *ATxmega128a1. Microchip [online]. USA, Arizona: Microchip, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.microchip.com/wwwproducts/en/ATxmega128a1.

13. *Katalog produktů atmel MCU automotive. USA, Arizona, 2013.* Dostupné také z: http://ww1.microchip.com/downloads/en/DeviceDoc/4622l_autosolutionsbrochure_e_us_103013_web.pdf.

14. *ATxmega128a1 MCU. In: Microchip [online]. USA, Arizona: Microchip, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.microchip.com/_images/products/medium/9cf66f35699220c89f32606711026d2c.png.

15. *Datasheet PIC18FXX8. USA, Arizona, 2004.* Dostupné také z: https://www.tme.eu/Document/b023ce06bde118f3a6c1c6bae01798ab/pic18fxx8.pdf.

16. *PIC MCUs. EL-PRO-CUS [online]. Indie - Hyderabad: ElProCus Technologies Pvt, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.elprocus.com/introduction-to-pic-microcontrollers-and-its-architecture/.

17. *PIC architektura. In: EL-PRO-CUS [online]. Indie - Hyderabad: ElProCus Technologies Pvt, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.elprocus.com/wp-content/uploads/2014/04/28.jpg.

18. *RH850 Renesas MCUs. Renesas [online]. Japonsko, Tokio: Renesas, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rh850-automotive-mcus.

19. *Nejvýkonnější MCU od Renesas. In: Renesas [online]. Japonsko, Tokio: Renesas, 2015 [cit. 2020-11-30].* Dostupné také z: https://www.renesas.com/sites/default/files/20150220-rh850-d1x_7.jpg.

20. *8051 MCU. EL-PRO-CUS [online]. Indie - Hyderabad: ElProCus Technologies Pvt, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.elprocus.com/pin-diagram-of-8051-microcontroller/.

21. *8051 architektura MCU. Tutorialspoint [online]. Indie, Telangana: Tutorials Point, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.tutorialspoint.com/microprocessor/microcontrollers_8051_architecture.htm.

22. *8051 architektura obrázek. In: Tutorialspoint [online]. Indie, Telangana: Tutorials Point, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.tutorialspoint.com/microprocessor/images/8051_architecture.jpg.

23. *DM system. ST [online]. USA: ST.com, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.st.com/en/applications/adas/driver-monitoring-system-dms.html.

24. *Katalog produktů ADAS. In: ST [online]. USA: ST.com, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.st.com/en/applications/adas/high-resolution-thermal-camera.html.

25. *Rear view camera. ST [online]. USA: ST.com, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.st.com/en/applications/adas/rear-view-camera.html.

26. *Mono camera. Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Cameras/Mono-Camera.

27. *Stereo camera. Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Cameras/Stereo-Camera.

28. *Multifunkční kamera. Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Cameras/Multi-Function-Camera-with-Lidar.

29. *SVS continental. Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Cameras/Surround-View-Camera.

30. *Mono camera obrázek. In: Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.continental-automotive.com/getattachment/9e64382c-da05-4df2-933d-7a6b86e50c38/attachment.aspx?width=500.

31. *Stereo kamera obrázek. In: Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].* Dostupné také z: https://www.continental-automotive.com/getattachment/3f97db37-1a86-40e2-8d6c-cdb148255f8a/attachment.aspx?width=500.

32. *Multifunkční kamera obrázek. In: Continental: The Future in Motion [online]. Německo: Continental, 2020 [cit. 2020-12-11].*

33. *Introduction to the LIN bus. KVASER [online]. Švédsko, Mölndal: Kvaser Europe AB, 2020 [cit. 2020-12-01].* Dostupné také z: https://www.kvaser.com/about-can/can-standards/linbus/.

34. *Schéma LIN přijímače. In: KVASER [online]. Švédsko, Mölndal: Kvaser Europe AB, 2020 [cit. 2020-12-01].* Dostupné také z: https://www.kvaser.com/wp-content/uploads/2014/02/fig1-lin-voltage-supply1.jpg.

35. *Rámeček zprávy LIN. In: KVASER [online]. Švédsko, Mölndal: Kvaser Europe AB, 2020 [cit. 2020-12-01].* Dostupné také z: https://www.kvaser.com/wp-content/uploads/2014/02/fig3-lin-frame-example1.jpg.

36. *Can protocol tutorial. KVASER [online]. Švédsko, Mölndal: Kvaser Europe AB, 2020 [cit. 2020-12-01].* Dostupné také z: https://www.kvaser.com/can-protocol-tutorial/.

37. *CAN bus. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-12-01].* Dostupné také z: https://en.wikipedia.org/wiki/CAN_bus.

38. *About CAN. KVASER [online]. Švédsko, Mölndal: Kvaser Europe AB, 2020 [cit. 2020-12-01].* Dostupné také z: https://www.kvaser.com/about-can/.

39. *CAN rámeček zprávy. In: KVASER [online]. Švédsko, Mölndal: Kvaser Europe AB, 2020 [cit. 2020-12-01].* Dostupné také z: https://staging.kvaser.com/wp-content/uploads/2014/01/2-canmsg-1-3.gif.

40. *FlexRay Automotive Communication Bus Overview. Engineer Ambitiously [online]. USB, Texas: NI, 2020 [cit. 2020-12-01].* Dostupné také z: https://www.ni.com/cs-cz/innovations/white-papers/06/flexray-automotive-communication-bus-overview.html.

41. *Statický segment FlexRay. In: Engineer Ambitiously [online]. USB, Texas: NI, 2020 [cit. 2020-12-01].* Dostupné také z: https://ni.scene7.com/is/image/ni/FlexRay_Static_Slots?scl=1.

42. *Example of cycle control. In: Engineer Ambitiously [online]. USA, Texas: NI, 2020 [cit. 2020-12-01].* Dostupné také z: https://ni.scene7.com/is/image/ni/FlexRay_in-cycle_control?scl=1.

43. *On-board diagnostics. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-12-01].* Dostupné také z: https://en.wikipedia.org/wiki/On-board_diagnostics#EOBD.

44. *OBD 2 konektor. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2014 [cit. 2020-12-01].* Dostupné také z: https://en.wikipedia.org/wiki/On-board_diagnostics#/media/File:OBD_connector_shape.svg.

45. *MOST bus. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020 [cit. 2020-12-01].* Dostupné také z: https://en.wikipedia.org/wiki/MOST_Bus.

46. *OpenCV platformy. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-17].* Dostupné také z: https://opencv.org/platforms/.

47. *Java Open source libraries ImageJ. Tutorials Point simply easy learning [online]. INDIE: Tutorials Point, 2021 [cit. 2021-03-17].* Dostupné také z: https://www.tutorialspoint.com/java_dip/open_source_libraries.htm.

48. *OpenCV dokumentace. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2020 [cit. 2021-03-17].* Dostupné také z: https://docs.opencv.org/master/d7/dbd/group__imgproc.html.

49. *OpenCV feature detectors. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-19].* Dostupné také z: https://docs.opencv.org/master/db/d27/tutorial_py_table_of_contents_feature2d.html.

50. *Harrisova detekce rohů. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-18].* Dostupné také z: https://docs.opencv.org/master/dc/d0d/tutorial_py_features_harris.html.

51. *Grafické vyjádření Harrisovy detekce rohů. In: Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-18].* Dostupné také z: https://docs.opencv.org/master/harris_region.jpg.

52. *Shi-Tomasův detektor rohů. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-18].* Dostupné také z: https://docs.opencv.org/master/d4/d8c/tutorial_py_shi_tomasi.html.

53. *Grafické vyjádření Shi-Tomasovy detekce rohů. In: Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-18].* Dostupné také z: https://docs.opencv.org/master/shitomasi_space.png.

54. *SIFT algoritmus. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-19].* Dostupné také z: https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html.

55. *Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). [cit. 2021-03-19].* Dostupné také z: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

56. *SURF algoritmus. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-19].* Dostupné také z: https://docs.opencv.org/master/df/dd2/tutorial_py_surf_intro.html.

57. *M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha and P. Fua, "BRIEF: Computing a Local Binary Descriptor Very Fast,"in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 7, pp. 1281-1298, July 2012, doi: 10.1109/TPAMI.2011.222.*

58. *E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF,"2011 International Conference on Computer Vision, Barcelona, Spain, 2011, pp. 2564-2571, doi: 10.1109/ICCV.2011.6126544.*

59. *E. Rosten, R. Porter and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection,"in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 1, pp. 105-119, Jan. 2010, doi: 10.1109/TPAMI.2008.275.*

60. *Obrázek FAST algoritmu. In: Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-19].* Dostupné také z: `https://docs.opencv.org/master/fast_speedtest.jpg`.

61. *ORB algoritmus. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-19].* Dostupné také z: `https://docs.opencv.org/master/d1/d89/tutorial_py_orb.html`.

62. *Porovnávání deskriptorů v OpenCV. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-19].* Dostupné také z: `https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html`.

63. *Basic concepts of the homography. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-23].* Dostupné také z: `https://docs.opencv.org/master/d9/dab/tutorial_homography.html`.

64. *Planar Homographies. USA, 2021.* Dostupné také z: `http://www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf`.

65. *Grafická ukázka výpočtu homografické matice. Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-23].* Dostupné také z: `https://docs.opencv.org/master/homography_transformation_example2.jpg`.

66. *Ukázka spojení panoramatu. In: Open Source Computer Vision [online]. USA: Intel Corporation, Willow Garage, Itseez, 2021 [cit. 2021-03-23].* Dostupné také z: `https://docs.opencv.org/master/homography_transformation_example3.jpg`.

67. *COLLINS, Robert. Planar Homographies obrázek str. 23. USA, 2021.* Dostupné také z: `http://www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf`.

68. *Mikrokontrolér Jetson Nano. NVIDIA developers [online]. USA: NVIDIA developers, 2021 [cit. 2021-04-06].* Dostupné také z: `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`.

69. *USB 2.0 kamera. Waveshare Wiki [online]. Čína: Waveshare, 2020 [cit. 2021-04-04].* Dostupné také z: https://www.waveshare.com/wiki/IMX179_8MP_USB_Camera_(A).

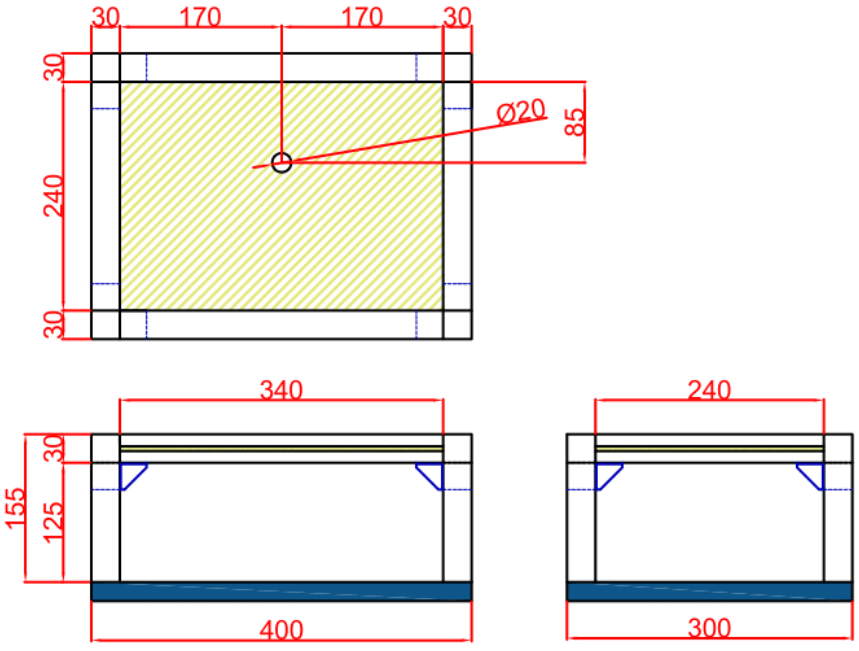
70. *USB 3.0 kamera. Aliexpress [online]. Čína: ELP Camera Module Factory Store, 2021 [cit. 2021-04-04].* Dostupné také z: https://www.aliexpress.com/item/32880997336.html.

71. *Kulový kloub. In: Plynovepruzinyshop [online]. Praha: plynove pruziny shop, 2020 [cit. 2020-11-30].* Dostupné také z: https://www.plynovepruzinyshop.cz/wp-content/plugins/gasveer_tools_serverside/img/caddrawings/WG30.png.

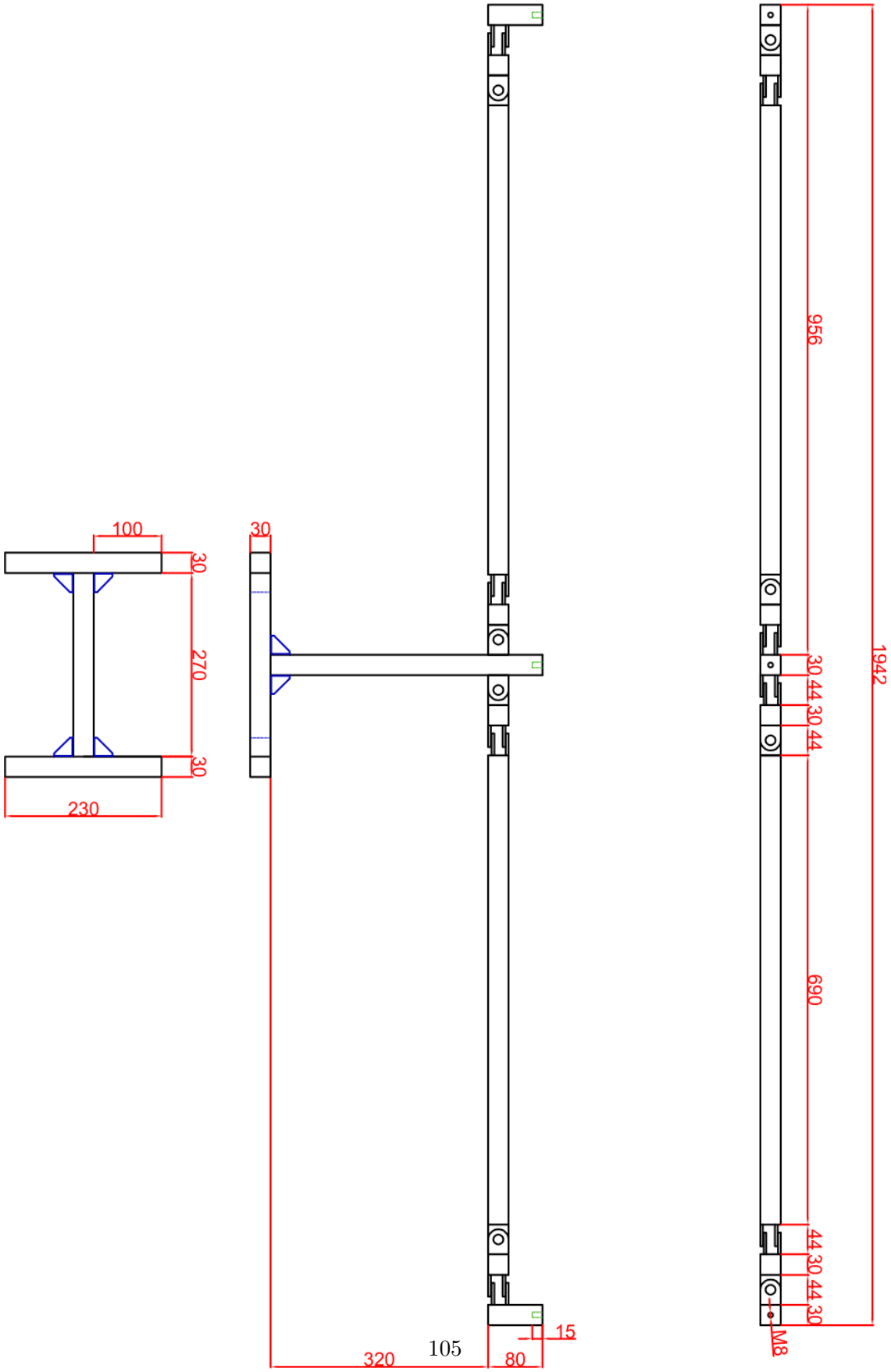
72. *KIRILLOV, Andrew. Cam2Web aplikace. Code project: For those who code [online]. Canada: CodeProject Submissions, 2021 [cit. 2021-03-26].* Dostupné také z: https://www.codeproject.com/Articles/1199725/cam-web-Streaming-camera-to-web-as-MJPEG-stream.

Příloha A

Výkresy stojanu a boxu



Obrázek A.1: Výkres boxu pro umístění výpočetní techniky



Obrázek A.2: Výkres stojanu reprezentující nárazník

Příloha B

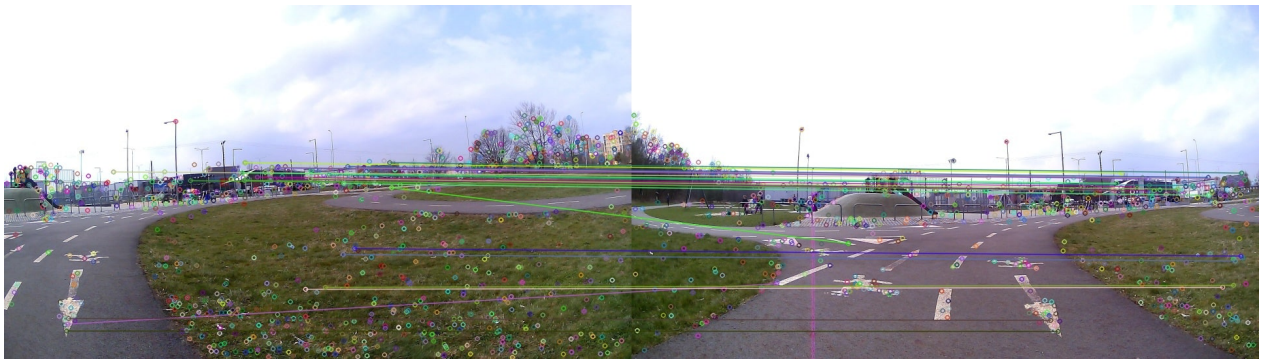
Vyhodnocení panoramatu č. 75 - výborný

Výsledný panoramatický snímek je sestaven ze tří zkreslených snímků s efektem rybí oko. Vizuální hodnocení je výborný. Mezi filtrovanými shodami nejsou žádné odchylky.

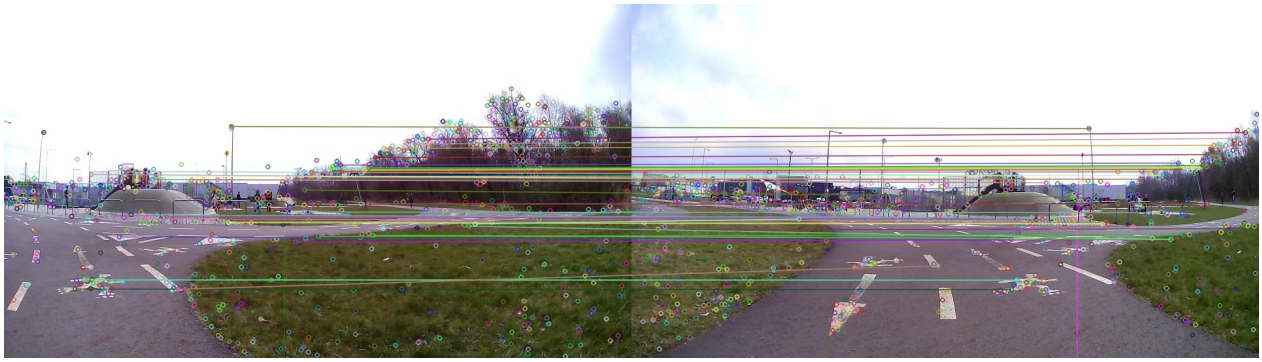
	Medián	RMS
Levá strana	12	20,091
Pravá strana	4	17,950

Tabulka B.1: Výsledné hodnoty RMS a mediánů panoramatu č. 75

(a) Panoramatický snímek č. 75

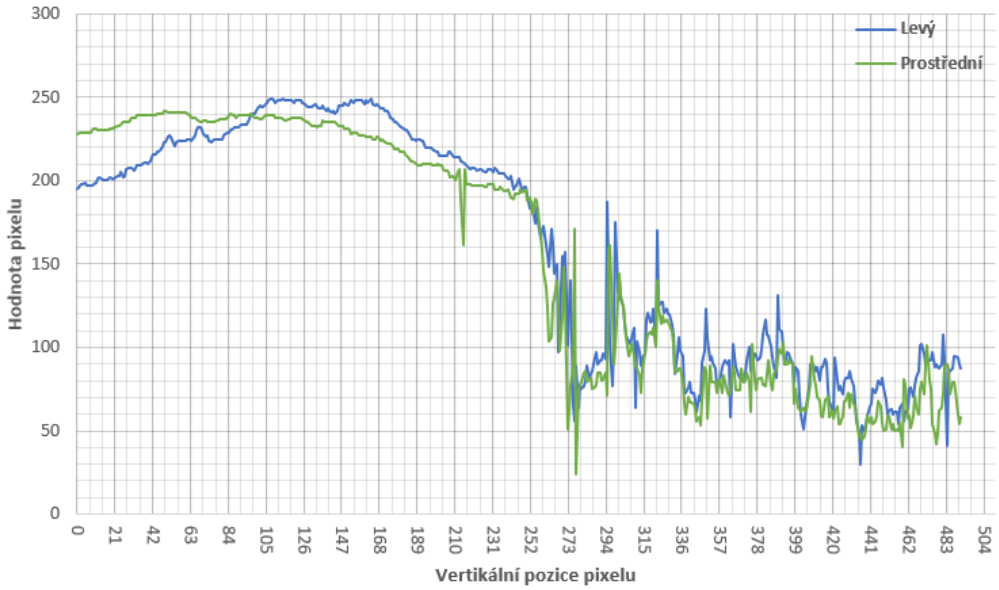


(b) Filtrované shody, levá strana

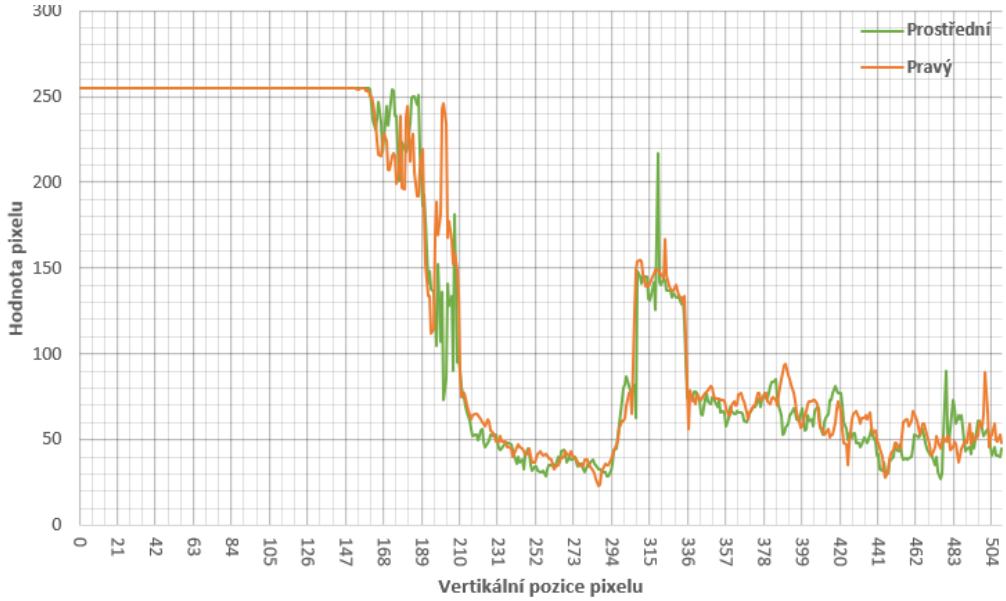


(c) Filtrované shody, pravá strana

Obrázek B.1: Panoramatický snímek č. 75 a nalezené shody

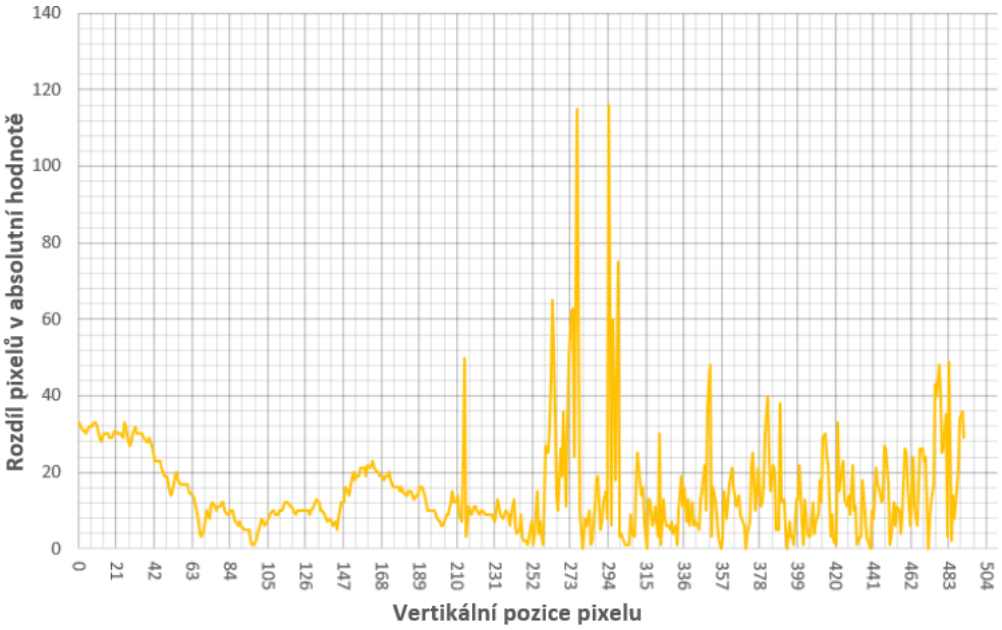


(a) Vykreslení hodnot, levá strana

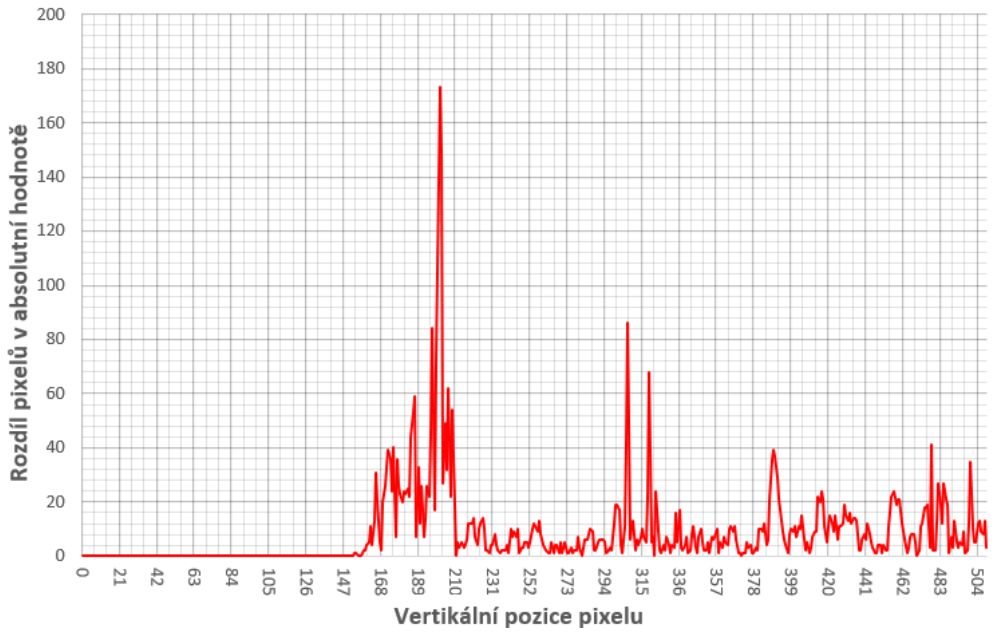


(b) Vykreslení hodnot, pravá strana

Obrázek B.2: Vykreslení souboru hodnot pixelů pro levou a pravou část panoramatu č. 75



(a) Rozdíl hodnot pixelů, levá strana



(b) Rozdíl hodnot pixelů, pravá strana

Obrázek B.3: Vykreslení rozdílů hodnot pro levou a pravou část panoramatu č. 75

Příloha C

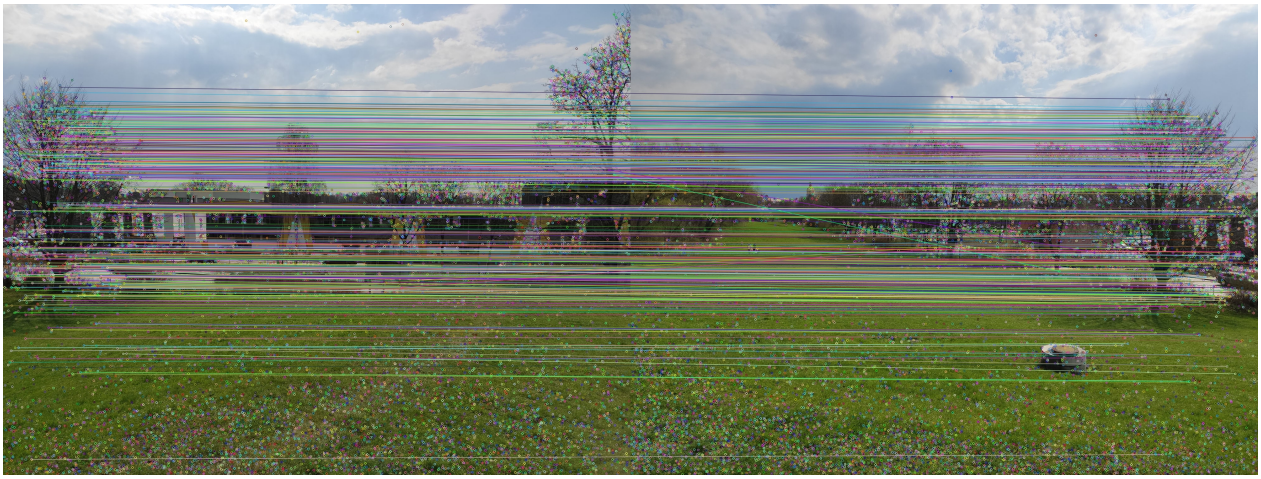
Vyhodnocení panoramatu č. 100 - výborný

Výsledný panoramatický snímek je sestaven ze tří lineárních snímků. Vizuální hodnocení je výborný. Mezi filtrovanými shodami je pár odchylek. Výsledné panorama obsahuje drobné chyby.

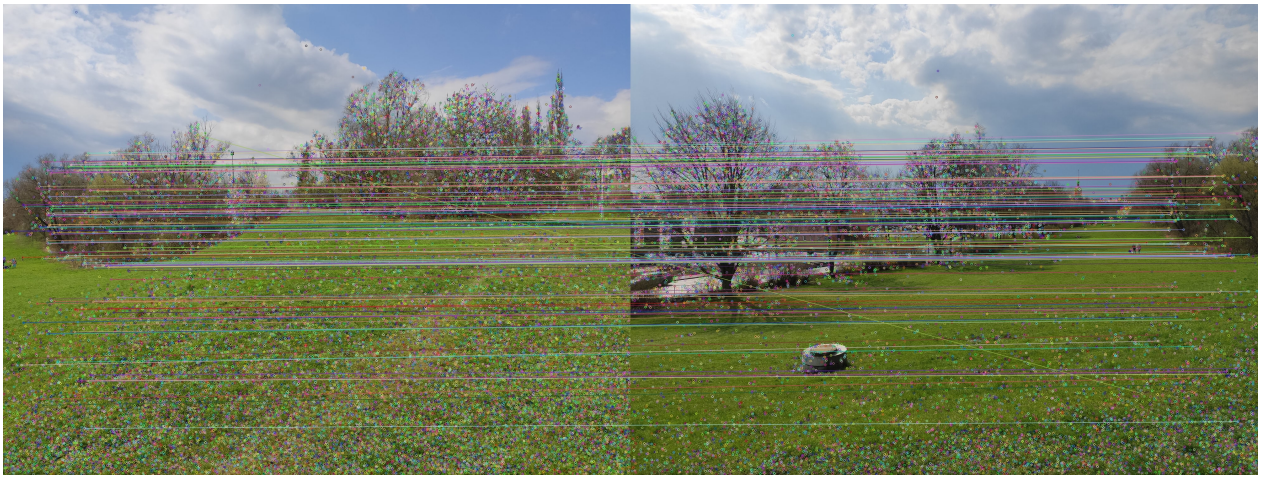
	Medián	RMS
Levá strana	7	22,152
Pravá strana	29	32,405
Levá strana (korekce jasu)	6	21,097
Pravá strana (korekce jasu)	17	22,697

Tabulka C.1: Výsledné hodnoty RMS a mediánů panoramatu č. 100

(a) Panoramatický snímek č. 100

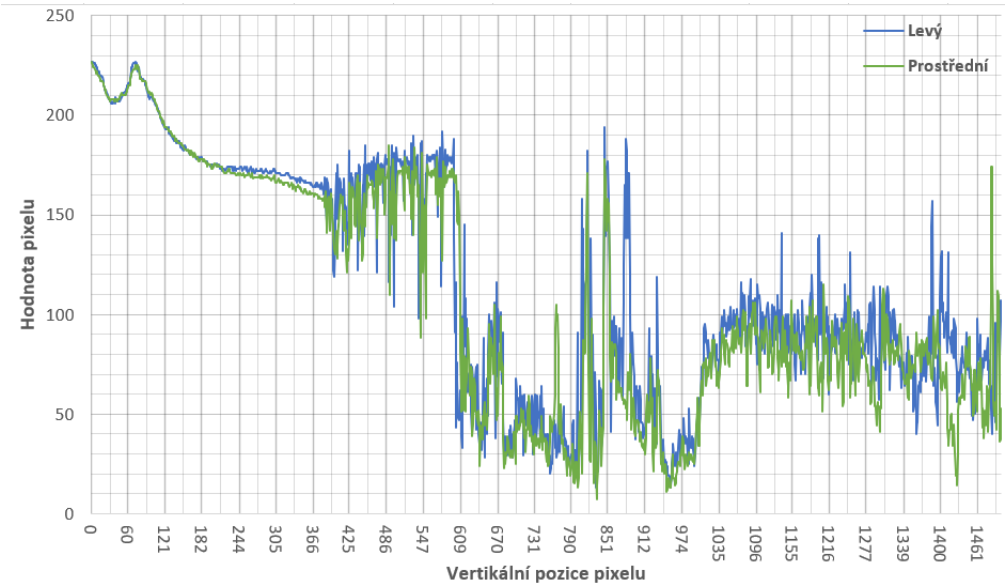


(b) Filtrované shody, levá strana

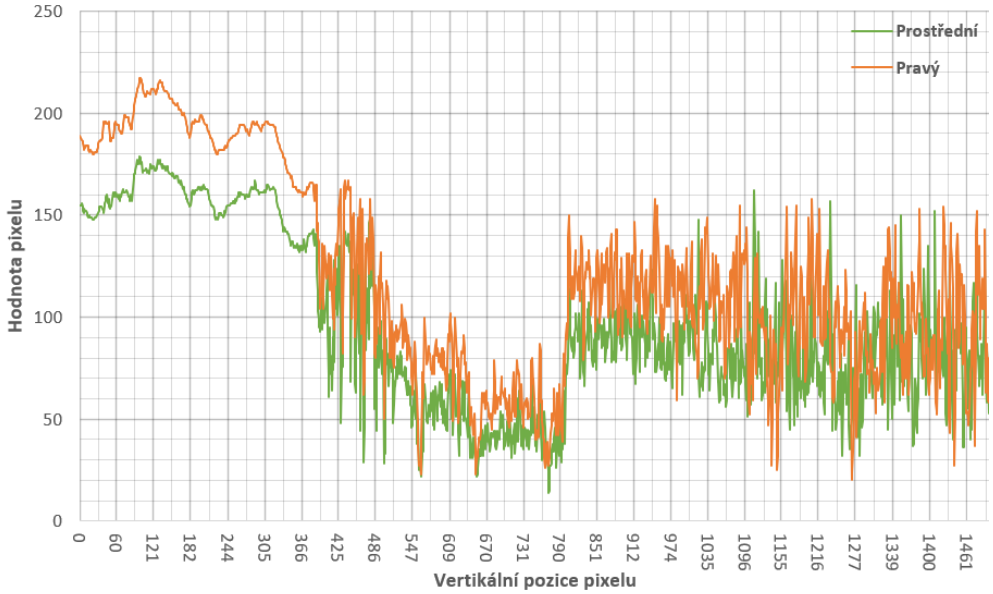


(c) Filtrované shody, pravá strana

Obrázek C.1: Panoramatický snímek č. 100 a nalezené shody

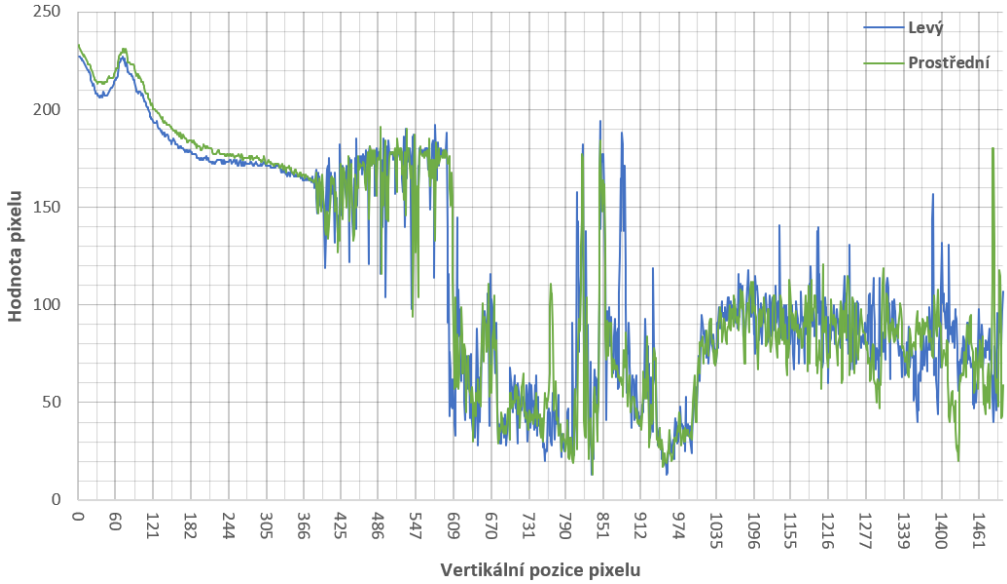


(a) Vykreslení hodnot, levá strana

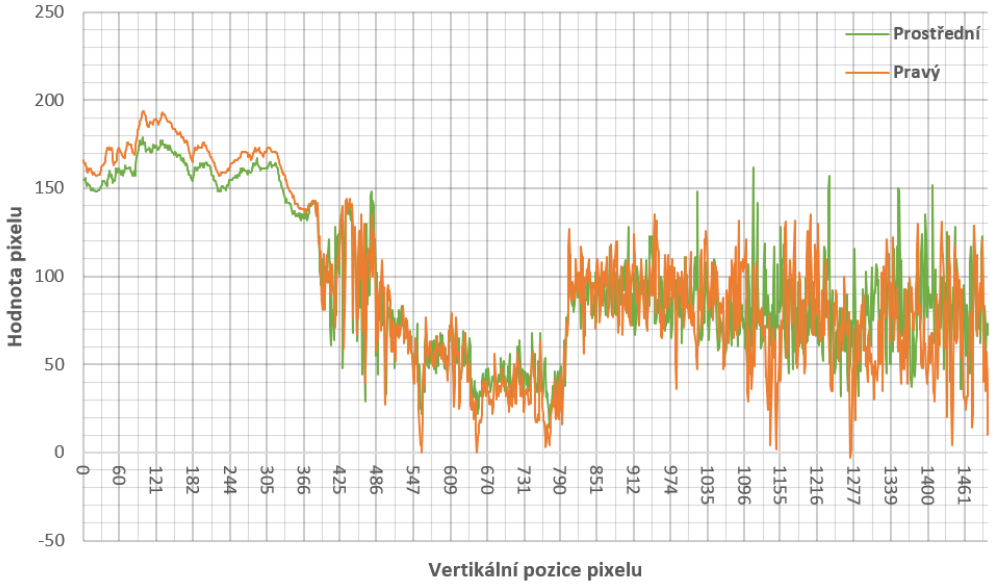


(b) Vykreslení hodnot, pravá strana

Obrázek C.2: Vykreslení souboru hodnot pixelů pro levou a pravou část panoramatu č. 100

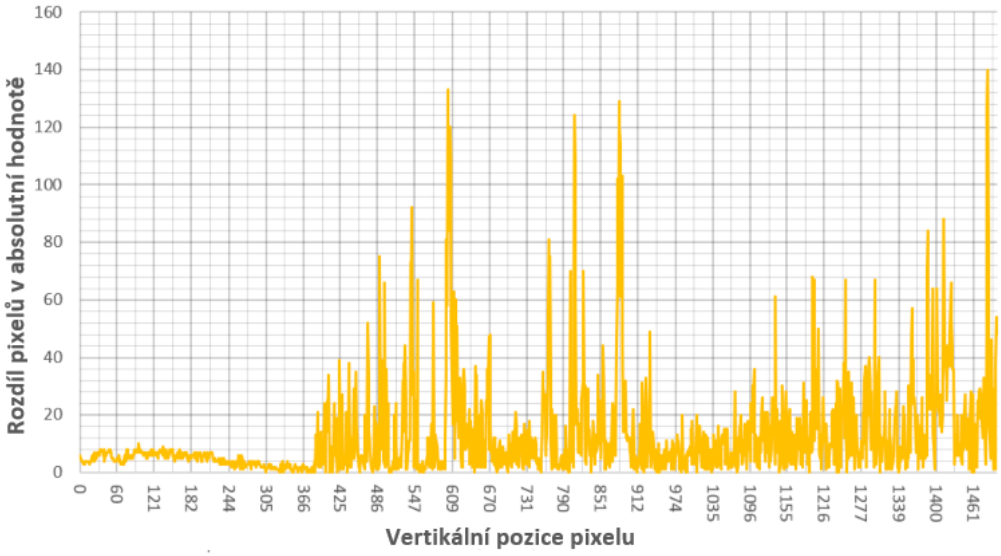


(a) Vykreslení hodnot s vyvážením jasu, levá strana

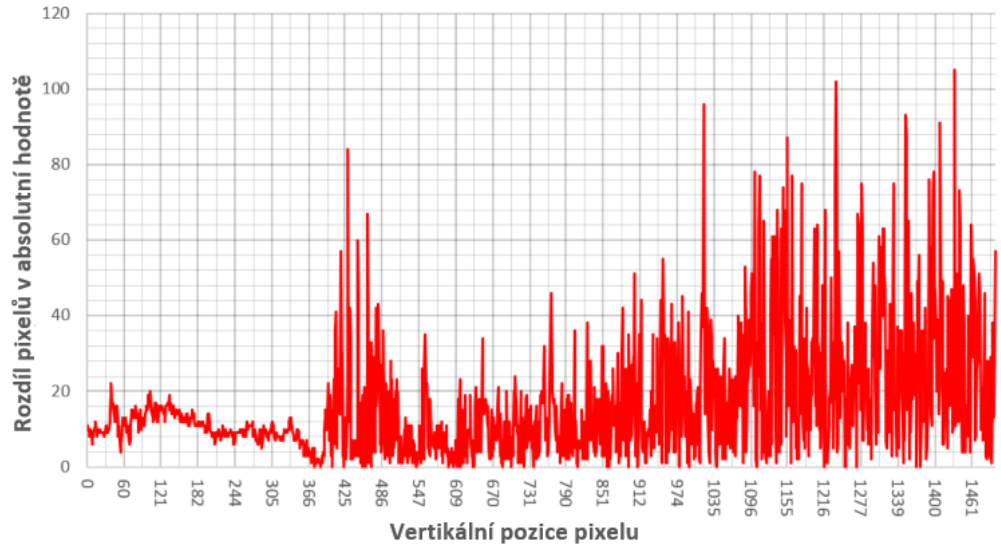


(b) Vykreslení hodnot s vyvážením jasu, pravá strana

Obrázek C.3: Vykreslení souboru hodnot pixelů po vyvážení jasu panoramatu č. 100



(a) Rozdíl hodnot pixelů s vyvážením jasu, levá strana



(b) Rozdíl hodnot pixelů s vyvážením jasu, pravá strana

Obrázek C.4: Vykreslení rozdílů hodnot pro levou a pravou část panoramatu č. 100

Příloha D

Vyhodnocení panoramatu č. 3 - přijatelný

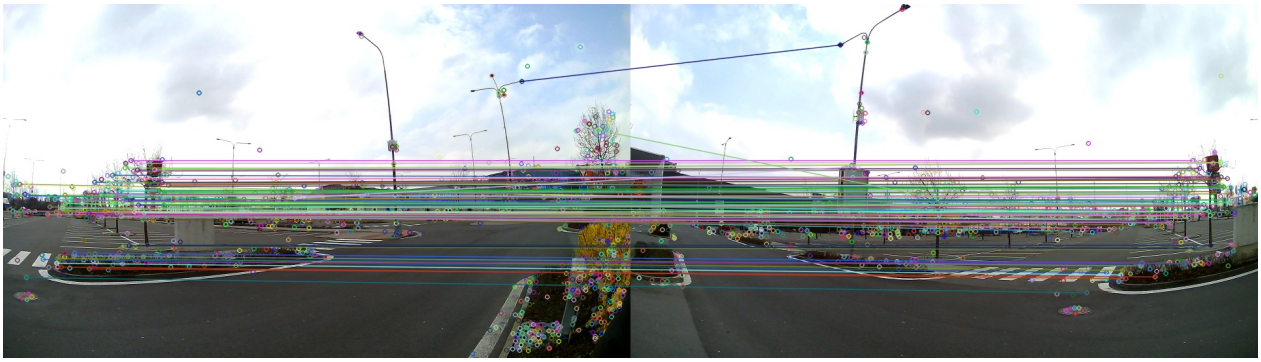
Výsledný panoramatický snímek je sestaven ze tří zkreslených snímků s efektem rybí oko. Vizuální hodnocení je přijatelný. Mezi filtrovanými shodami nejsou žádné odchylky.

	Medián	RMS
Levá strana	8	46,463
Pravá strana	8	27,578

Tabulka D.1: Výsledné hodnoty RMS a mediánů panoramatu č. 3

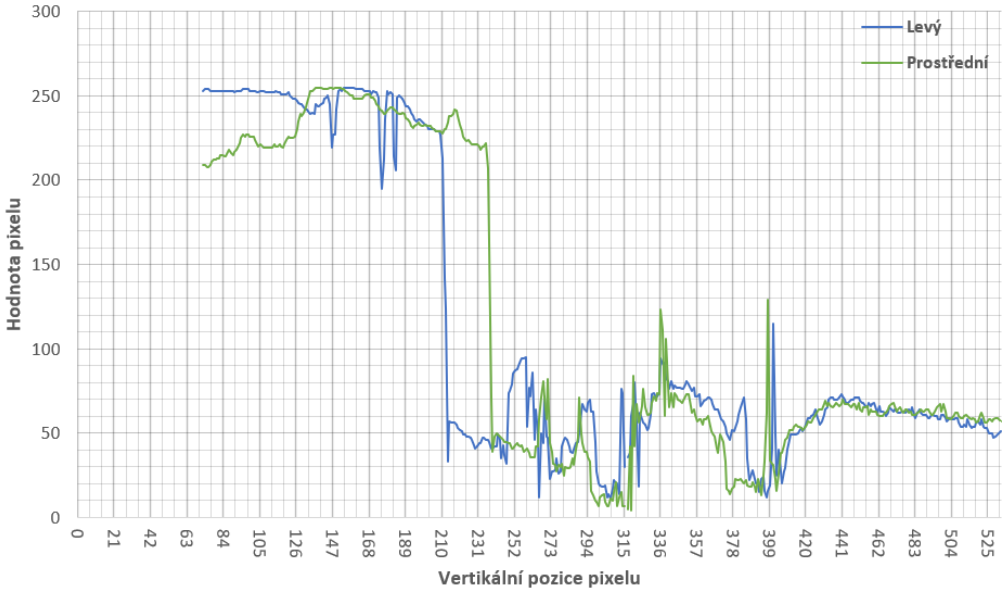
(a) Panoramatický snímek č. 3

(b) Filtrované shody, levá strana

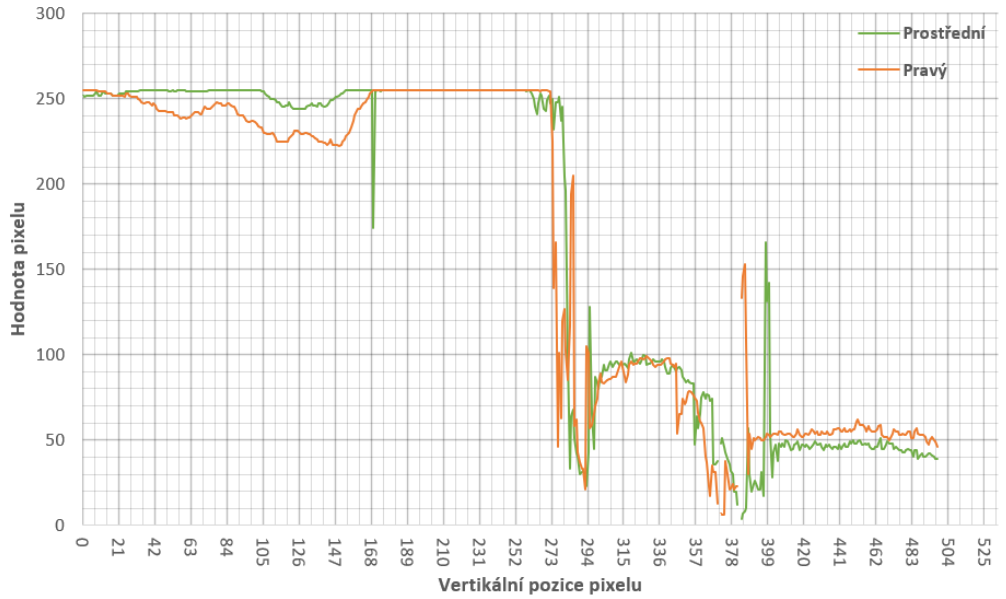


(c) Filtrované shody, pravá strana

Obrázek D.1: Panoramatický snímek č. 3 a nalezené shody

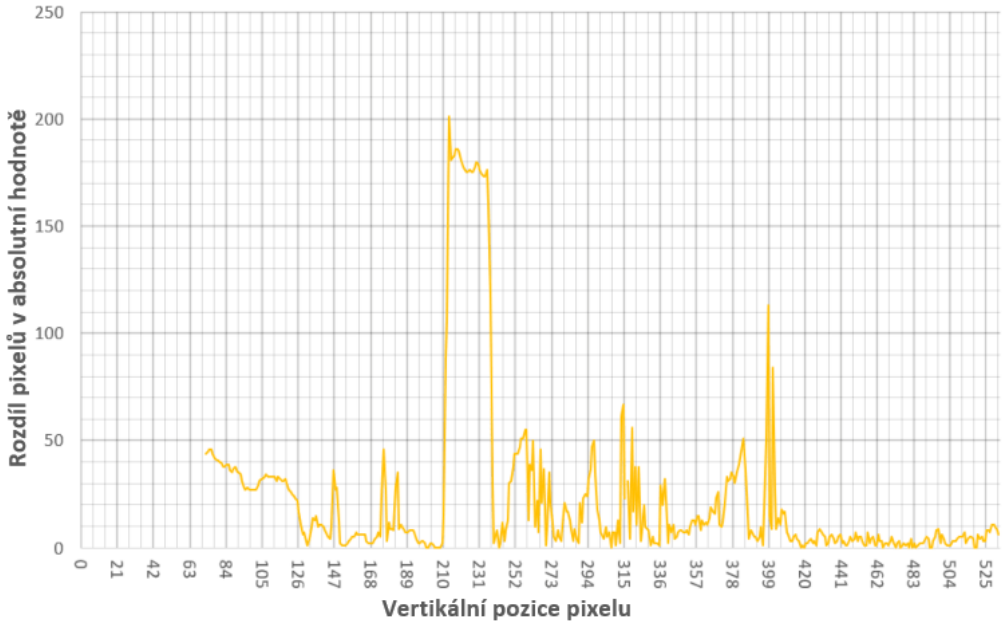


(a) Vykreslení hodnot, levá strana

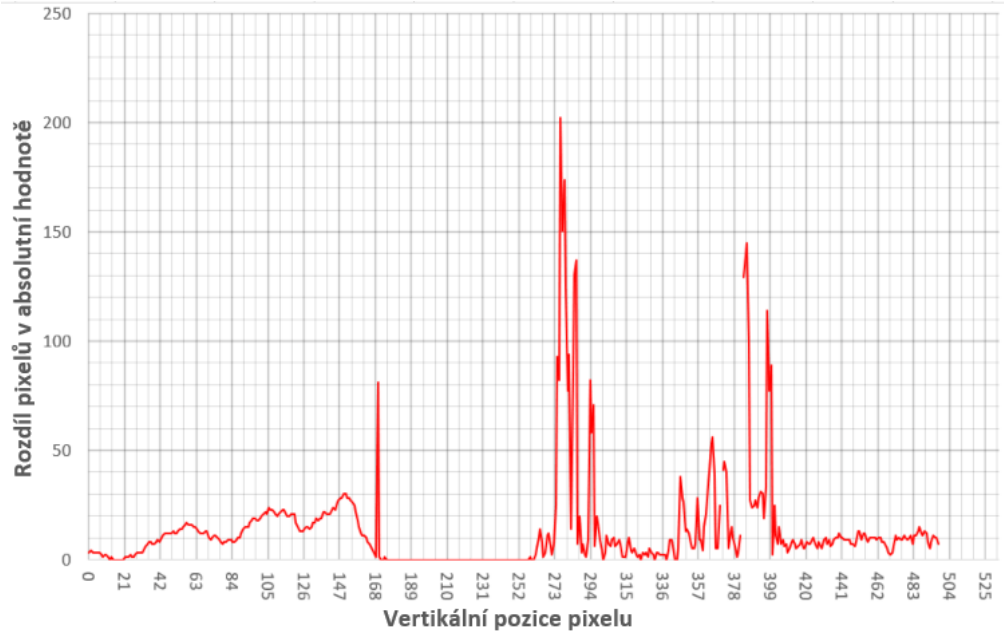


(b) Vykreslení hodnot, pravá strana

Obrázek D.2: Vykreslení souboru hodnot pixelů pro levou a pravou část panoramatu č. 3



(a) Rozdíl hodnot pixelů, levá strana



(b) Rozdíl hodnot pixelů, pravá strana

Obrázek D.3: Vykreslení rozdílů hodnot pro levou a pravou část panoramatu č. 3

Příloha E

Vyhodnocení panoramatu č. 22 - nedostatečný

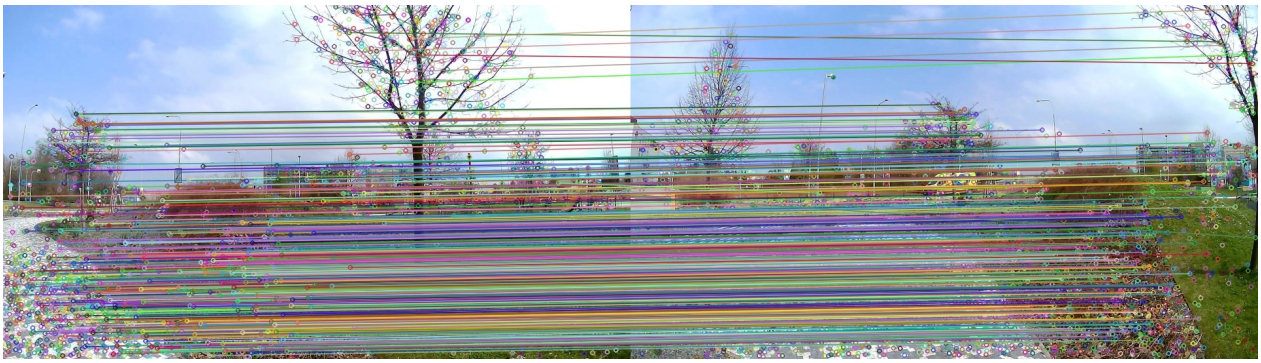
Výsledný panoramatický snímek je sestaven ze tří zkreslených snímků s efektem rybí oko. Vizuální hodnocení je nedostatečný. Mezi filtrovanými shodami je několik desítek odchylek.

	Medián	RMS
Levá strana	21	42,723
Pravá strana	63	120,48

Tabulka E.1: Výsledné hodnoty RMS a mediánů panoramatu č. 22

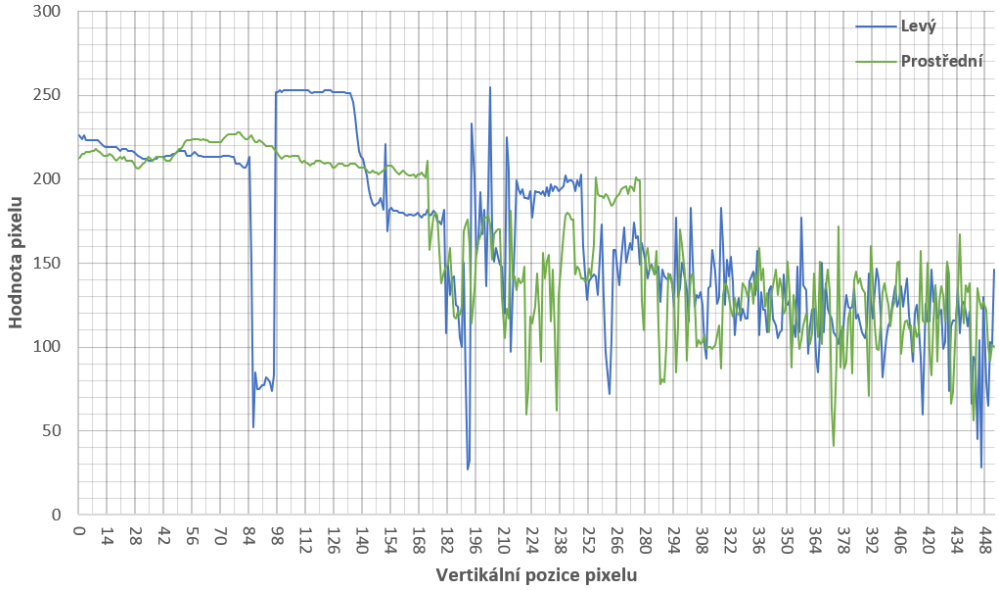
(a) Panoramatický snímek č. 22

(b) Filtrované shody, levá strana

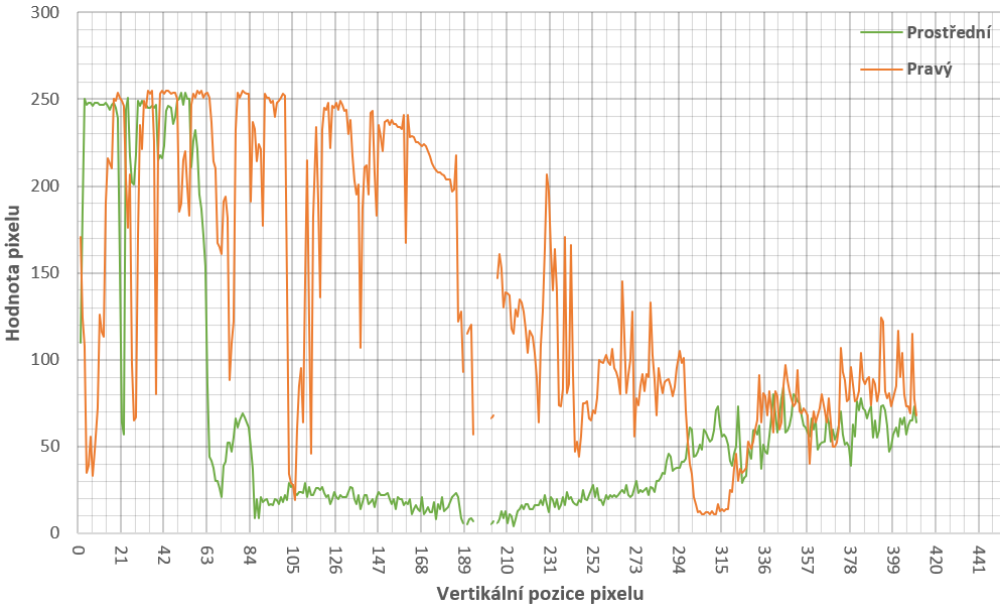


(c) Filtrované shody, pravá strana

Obrázek E.1: Panoramatický snímek č. 22 a nalezené shody

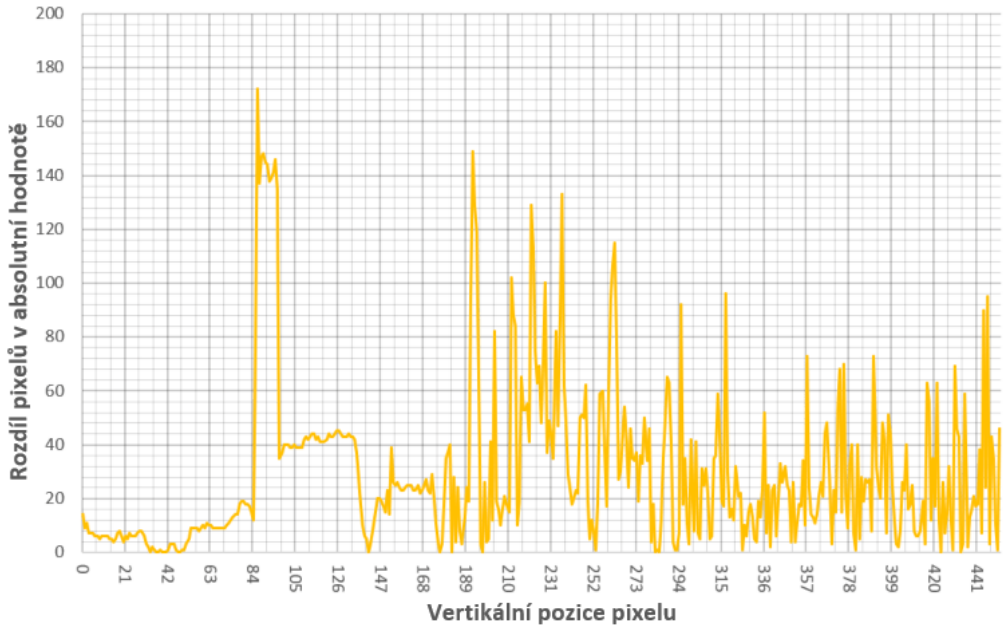


(a) Vykreslení hodnot, levá strana

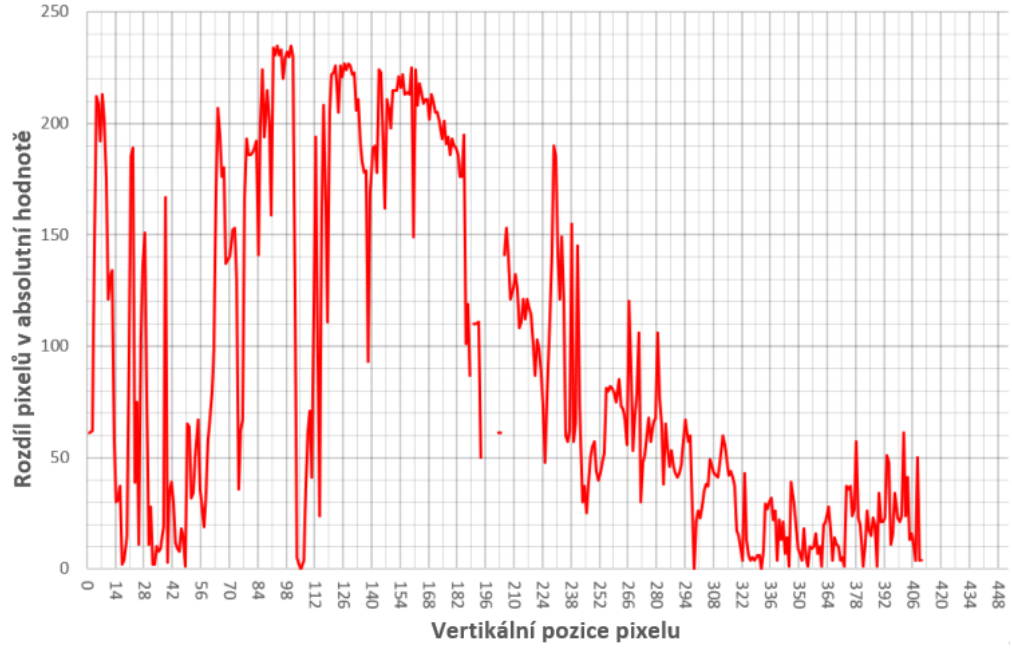


(b) Vykreslení hodnot, pravá strana

Obrázek E.2: Vykreslení souboru hodnot pixelů pro levou a pravou část panoramatu č. 22



(a) Rozdíl hodnot pixelů, levá strana



(b) Rozdíl hodnot pixelů, pravá strana

Obrázek E.3: Vykreslení rozdílů hodnot pro levou a pravou část panoramatu č. 22